THE BEST SELLING COMPUTER PROJECTS MAGAZINE DECEMBER 1984

ELECTRONICS & COMPUTING

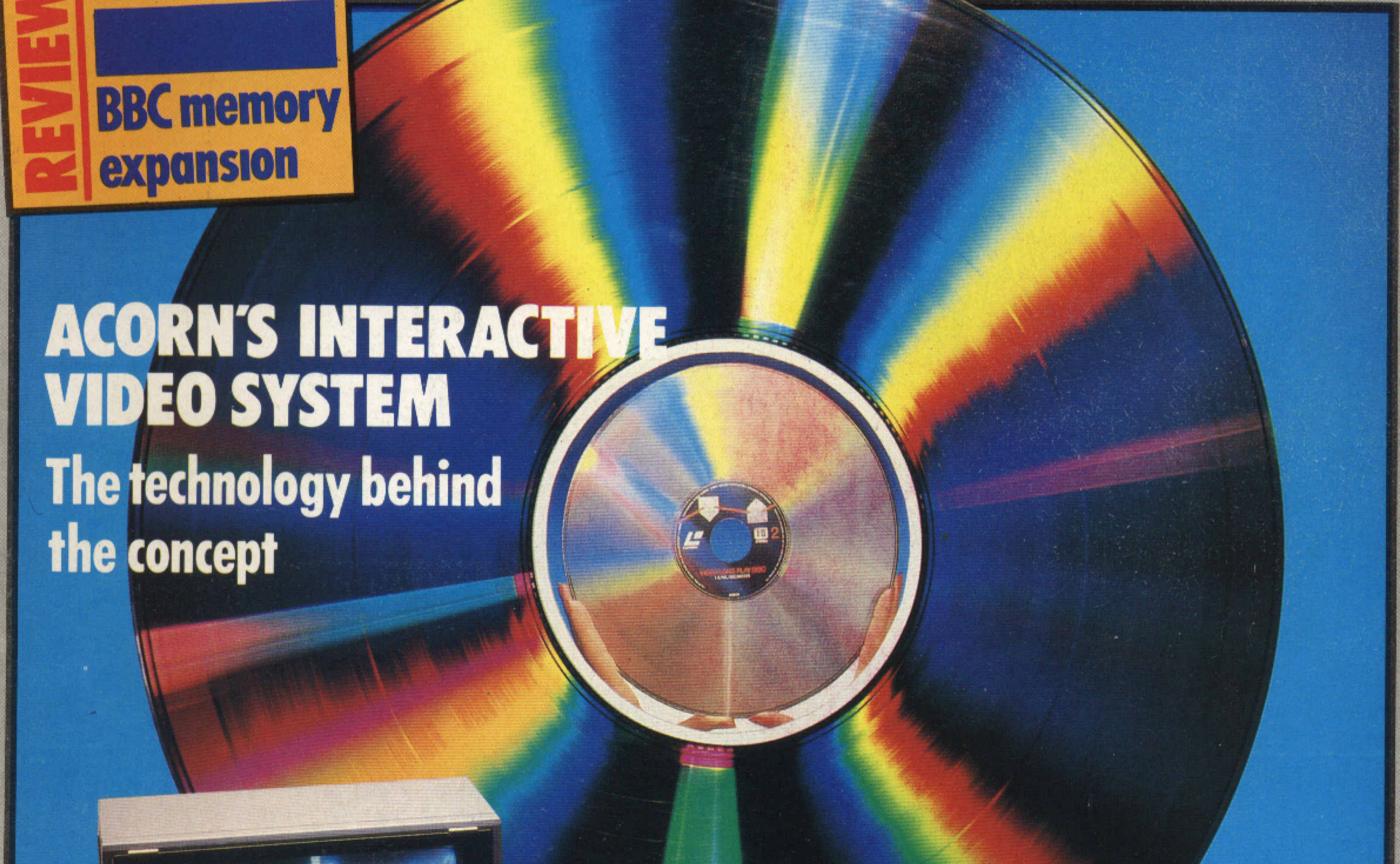
9 8255 CHIP x CAT
14/16 PRINTER
22 41 CHECK FOR IC FOR DIG. CASSETTE
42 51 DISC MOD
58 INFO
60 66 REF
ROBOT 71 CHECK FOR OWN UNIT
88 ORIC HARDWARE ETC

AN EMAP PUBLICATION

USA $2.95 Germany D6.00 95p

REVIEWS Spectrum +

BBC memory expansion



ACORN'S INTERACTIVE VIDEO SYSTEM
The technology behind the concept

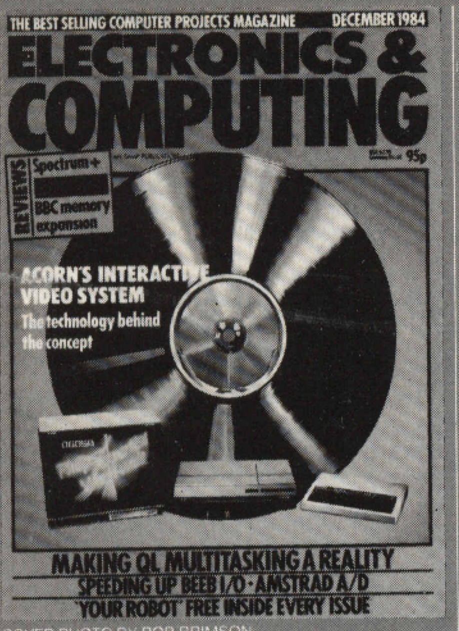
MAKING QL MULTITASKING A REALITY

SPEEDING UP BEEB I/O · AMSTRAD A/D

'YOUR ROBOT' FREE INSIDE EVERY ISSUE

ELECTRONICS & COMPUTING Contents

Vol. 4 Issue 12


COVER PHOTO BY ROB BRIMSON

Electronics & Computing Monthly
Scriptor Court, 155 Farringdon Road,
London, EC1R 3AD

Editorial 01-833-0846
Editor Gary Evans
Deputy Editor William Owen
Production Editor Liz Gregory
Administration Serena Hadley

Advertising 01-833-0531
Advertisement Manager Anthony Herman
Chief Executives Richard Jansz
Philip Baker
Classified Tracey Keighley
Advertising Production Yvonne Moyser

Production 01-833-0531
Art Editor Jeremy Webb
Make-up Time Graphics
Publisher Alfred Rolington

Distribution
EMAP National Publications

Published by
EMAP Business and
Computer Publications

Printed by
Riverside Press, England

Subscriptions
Electronics & Computing Monthly,
(Subscription Department),
Competition House, Farndon Road,
Market Harborough, Leicestershire.

Electronics & Computing Monthly is normally published on the 13th day of each month.

ABC

Computing with Burke's

In its 158 year history, Burke's Peerage, the reference book detailing the blood lines of the British upper classes, has got along quite well without any help from the computer. Next year's edition of the work will, though, make use of the latest computer hardware to collate all the information going into the issue. The move onto a computerised system has, however, given Burke's quite a few headaches.

It seems that the publishers of Burke's had expected rather too much of the humble computer which, in the words of a company spokesman, can only do what they are told. It appears as if, in their innocence, the people at Burke's were not aware of this fact.

The problems with the computer system mean that there will be a couple of months delay in publication of the issue, something that is bound to hurt the pockets of the publishers.

One burke is quoted as saying that in the old days the title was put together by an army of quill-wielding clerks and that as far as he was concerned, the quill was still mightier than the computer – I hope his Managing Director shares that opinion.

Bevan's Beeb Drive

Bevan Technology, a company new to the micro electronics marketplace, has chosen a 3" disk drive for the BBC micro as their product. The drive is based on a mechanism manufactured by Hitachi and is capable of storing 100K on each side of the flippable mini-floppy disks.

Each drive is supplied with a systems disk that includes many utilities in addition to a memory mover routine and a screen dump facility. The command screens of the utilities disk make full use of the sound and graphics pages of the BBC computer.

Prices of the products range from £159 for a single sided, single drive unit to £359 for a dual, double sided unit.

Bevan have plans to extend their range of BBC micro add-ons early in the new year. At present prospects include a comprehensive interface that builds in a 3" drive plus a range of I/O functions. The unit is to be housed in a casing that will bridge the computer thus acting as a convenient base for a monitor.

Bevan Technology Limited, 3rd Floor, Gresham Chambers, 14 Lichfield Street, Wolverhampton, West Midlands, WV1 1DG.

Acorn Competition

Rumours that Acorn are to launch a new computer onto the home computer market in the first quarter of next year are beginning to gain momentum. Two independent sources have suggested to *E&CM* that the company have plans for a spring launch although so far details of the new product have proved rather scarce.

From the commercial point of view, it must make sense for Acorn to re-examine their position in the home computer market before they lose the key position that they hold at the present time. The Electron was to have been the computer to consolidate the company's position in the home field but for various reasons this machine has not caught the buying public's imagination and shows no sign of attaining the aspirations that Acorn must at one time have held for the machine. Another consideration is that, while the new range of business micros represents a new and important step for Acorn, there are many that doubt the wisdom of pinning too many hopes on hardware that begins to compete with the IBMs of this world. And although, Acorn are at present a big fish in the UK home computer market, they are but a drop in the proverbial in terms of the worldwide business computer market.

If Acorn are to continue to be as successful as they have been over the past few years it is essential that the company consolidate their position and come up with a micro that will carry on from where the ageing BBC micro leaves off. For this reason any new computer will be vital to the continued success of the company.

Details of the new machine are, as we have said, scarce and so we are able to say little of what the new computer will offer. The rumours indicate anything from a minimal re-working of the established design that would make provision for the inclusion of a suite of software as in Commodore's new Plus4 computer to a QL-like 16 bit computer that offers full compatibility with BBC Basic.

What do you think? If you were Chris Curry what design brief would you give to the Acorn R&D labs? Bearing in mind that the future success of your company could well depend on coming up with the right specification what would you do? We would like to hear your thoughts on this matter. Write down, on a postcard please, the five most important elements in your view as to what Acorn should be doing. For example, should they opt for a 16 bit processor and if so, which one? Should the operating system be Unix, CP/M 68K or any of the other established OSs or should Acorn, like Sinclair come up with their own software? What about price?

We'll be selecting the most interesting specifications in our March issue and anyone whose thoughts appear in print will receive a bottle of bubbly (non vintage unfortunately!) courtesy of *E&CM*. As the postal service will begin to degenerate with the run up to Christmas in the coming week or so, get your thoughts together as quickly as possible and send your postcards to:

Acorn Competition, Electronics and Computing, Scriptor Courty, 155 Farringdon Road, London, EC1R 3AD.

Patently obvious for Aries?

Just after the last issue of *E&CM* closed for press, Cambridge Computer Consultants published their patent application in respect of their Aries B20 memory expansion board for the BBC micro. We are at present taking advice from a number of bodies as to the likelihood that the Aries patent would in fact be granted. It is hoped that, in our January issue, we will be able to report fully on the content and scope of the patent application and the reaction to both from the technical and legal point of view.

We want you!

E&CM are looking for an electronics enthusiast to assist in the testing and researching of projects for future inclusion in the magazine.

Suitable applicants will have had some electronics experience and may possibly have contributed to magazines in the past. Any expenses incurred during the building up of projects will be met by *E&CM* and the magazine will also be offering a small fee which will be paid on a freelance basis.

For convenience, all applicants should live in the London area, preferably in the south, or east or in the city so that contact with the staff on the magazine may be made without too much difficulty.

Anyone interested in getting involved with *E&CM* at this level should apply to The Projects Editor, Electronics and Computing Monthly, Scriptor Court, 155 Farringdon Road, London EC1R 3AD.

New printers from Quen-Data

Those considering the purchase of a printer this Christmas might well be advised to wait until late December when Quen-Data, the UK distributor for Sumitomu, the largest manufacturer of computer peripherals, are to launch four new models. These will have similar specifications to the Epson range of printers but will be available at a much lower cost.

In the meantime, Quen-Data are offering two printers of interest to home users. These are the DWP1120 daisy wheel printer retailing at £280 and the DP100 dot matrix unit available for £230. The models are available from Computer Intelligence, Spectrum UK and many other major High Street distributors.

Prospects for Projects

In our October issue we mentioned the fact that we intended to give away a free copy of Sinclair Projects with last month's *E&CM*. Unfortunately, due to circumstances beyond our control, we were unable to complete production of the magazine as planned.

In future we will be incorporating Sinclair Projects within *E&CM* and have lined up some exciting projects for the Spectrum, QL and ZX81. These will begin to appear early in the new year, the first in our February issue. We hope that past readers of Sinclair Projects will find plenty of material to interest them within the pages of *E&CM*.

NEWS NEWS NEWS NEWS

Kempston's Double Joy

Kempston Micro Electronics have launched two new joysticks just in time for the Christmas games boom. A move away from the Spectrum market brings the BBC Pro model, which costs £16.95 inclusive of VAT and is designed to blend in with the micro itself plugging directly into the analogue port whilst the Junior Pro is aimed at Spectrum owners and carries a price tag of £5.99.

The company are aiming to boost sales further with a special Christmas package of Junior Pro joystick and Kempston interface on offer at £14.99, which is a saving of £2.50 if the items are to be bought separately. Kempston are at Singer Way, Woburn Road Industrial Estate, Kempston, Bedford, MK42 7AF.

Wise-up on Wordcraft

Our series of wordprocessing reviews featured in last month's issue made mention of the CBM package, Wordcraft. However, we neglected to add the name and address of the suppliers to our chart, an omission for which we must apologise.

Wordcraft 64 (disk version) is priced at £43.44 plus VAT and is available from Dataview Wordcraft, Radix House, East Street, Colchester, Essex CO1 2XB.

Quest for QL add-ons

The QL add-on industry is starting to gain momentum and one of the most impressive entries in the field to date are Quest. The company recently unveiled a range of hardware and software for the new Sinclair computer. The bolt-on hardware goodies, designated the QL Executive Series, include a range of disk drives and interfaces which start from a humble 200K floppy drive and go to a 7.5MB Winchester.

The major software announcement concerns the company's implementation of digital Research's CP/M 68K OS in a QL compatible form. This is supplied either on a 5¼" floppy disk (price £49.50) or on microdrive cartridges (£79.50). The 28.5K operating system includes an assembler and goes alongside the QL's own QDOS operating system.

Quest have also produced a suite of business accounts software that goes under the name Tally. The packages in this range of software are integrated not only within themselves but with the four Psion packages that are supplied with the QL. Thus it is possible, for example, to transfer sales revenue figures from the nominal ledger program and to display them graphically using Easel.

The Quest add-ons take the QL to a level of performance that truly advances the system to the point at which it can fulfill a useful role in a small business application.

We hope to feature a more detailed look at the Quest range of products in the next month or so and, in particular, to examine the CP/M-68K OS and the way in which it interacts with QDOS.

Quest, School Lane, Chandler's Ford, Hampshire, SO5 3YY.

NOTE

Paul Beverley has asked us to point out that his article describing the use of the BBC micro as a Logic State Monitor, published last month, was based on an idea from Mr. G. G. Osbourne.

GST Assembler: a re-review

In the last issue we reviewed the GST QL assembler, saying that we did not think much of it. We also said that it didn't produce real error messages. We were wrong! After a more thorough examination of the assembler we felt that it was only fair to GST if we reviewed the product again. Remember that this is the product which Sinclair is going to sell under its own name for £39.95.

The assembler is supplied on a microdrive cartridge and includes a large and comprehensive manual. The cartridge contains a number of files – the assembler itself, an editor (the Metacomco one mentioned last month), a whole host of header files containing all the QDOS equates and offsets and a **boot** program which can be used to run the assembler if required.

The program is invoked by the EXEC command, and it first asks for a console definition. Typing ENTER alone sets the default window and then it prompts for a command line.

As we said in the last issue this prompt is a less friendly way of invoking the assembly, but it comes down to personal preference in the end.

All files to be assembled have the default extension of **_asm** and the binary files it produces are always given the extension **_bin.** This can at times be tedious as it has become a sort of de facto standard that QL files to be CALLed are terminated with **_code** and those which are intended to be EXECed are terminated with **_exec.** Although this does make it rather harder to follow this standard it makes the entering of a command line into the assembler that much easier. If we wanted to assemble a file called **mdv1_pipedemo_asm** all we would have to do is type in a command line like:

mdv1_pipedemo

as the default extensions will be used, resulting in a code file called **mdv1_pipedemo_bin** and a listing file called **mdv1_pipedemo_list.** Horses for courses.

The command line allows you to specify the listing file (if any) so that it is easy to send it to the printer. The error messages, which are by the way fully comprehensive, can be sent to a file by themselves if required.

Apart from its debatable unfriendliness the product turns out to be very professional as the assembly is very fast indeed and a symbol table is produced at the end which can be useful. The listing produced contains all the information that you would expect (line number, address, hex code, assembly instruction) plus a comprehensive page header which even tells you the time when the assembly was done!

In the event, the GST assembler follows the Motorola convention very closely, and even gives warnings in the light of certain occurrences. If, for example, you specified a long branch when in fact a short branch would have done, then a warning is given. It also appears that the short option is taken.

As programs in this magazine will, from time to time, use some of the QL assemblers available, it is worth noting that GST's DATA directive is equivalent to Metacomco's SIZE directive, and INCLUDE in the former is the same as GET in the latter.

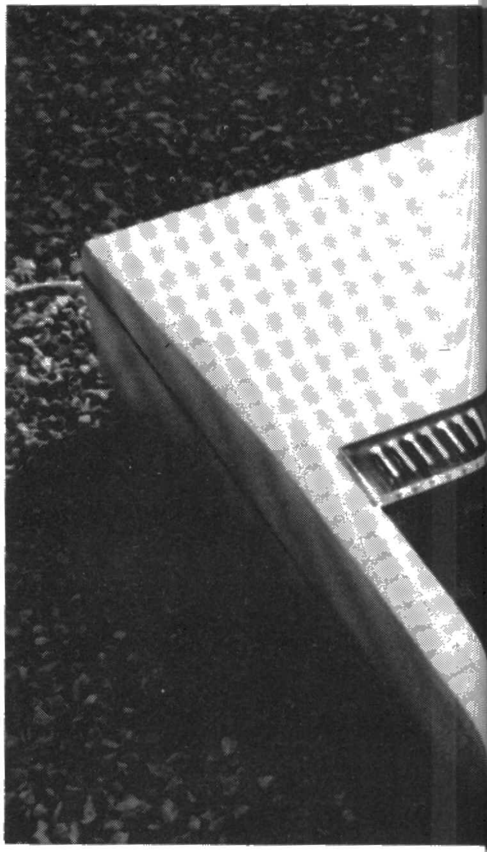
So, of the two assemblers which we have reviewed, which is the best? This is hard to say – the GST assembler is far more compact and allows for much larger assemblies, but the Metacomco product caters for macros and is rather more friendly. The price difference of £20 is significant too, as the GST assembler seems rather better value for money.

In the end all assemblers do roughly the same job, so it must be a compromise taken by the user. We use both here, depending on the circumstances.

I must apologise to GST for the inaccurate review of their assembler in the last issue – it was entirely my fault but was not intended to reflect any bias or malice.

Adam Denning

HIGH SPEED I/O



Paul Beverley shows how to interface the BBC micro using the full 2MHz clock speed of the 6502 processor.

The BBC microcomputer uses a 6502A processor, which means that it is capable of running at clock speeds of up to 2 MHz. This would normally mean that one would have to use interface chips which can also work at 2 MHz, such as the 6522A. The problem is that such chips are more expensive than their 1 MHz counterparts, therefore to get the higher processing speed and yet avoid the extra expense, the designers of the BBC microcomputer have made the system capable of operating at the speeds 2MHz for high speed processing and 1 MHz for slower speed interfacing.

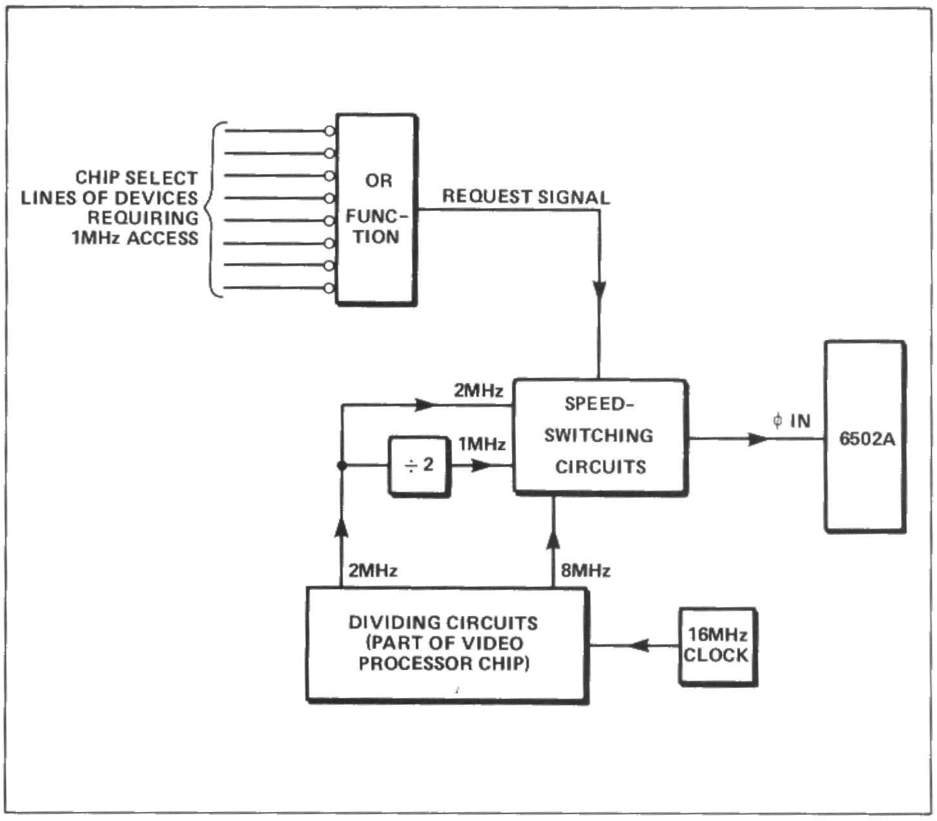
In this article we will be looking at the way in which this dual speed processing is achieved and trying to see how it affects the timing of interfacing to external devices. Then we shall look at ways in which the slower interfaces can be speeded up to 2 MHz in order to get the best out of the 6502A's high processing speed when trying to interface to high speed peripherals. Presumably, if you wish to increase the speed of the interfaces you must already have tried to do some programming in 6502 assembly language in order to have recognised the speed limitations. Therefore, in trying to explain what is happening, we will assume some knowledge of 6502 mnemonic codes.

The hardware

The hardware for this dual speed processing is illustrated in block diagram form in **Figure 1.** The idea is that the circuit automatically slows the processor down to 1 MHz every time it tries to access certain devices. To do this, the chip select lines for all the devices that are to operate at slow speed are gated together to form a "request" signal. This signal is then applied to a circuit comprised of a number of gates and bistables which sets up the timing of the signal which is fed into the clock input to the 6502A.

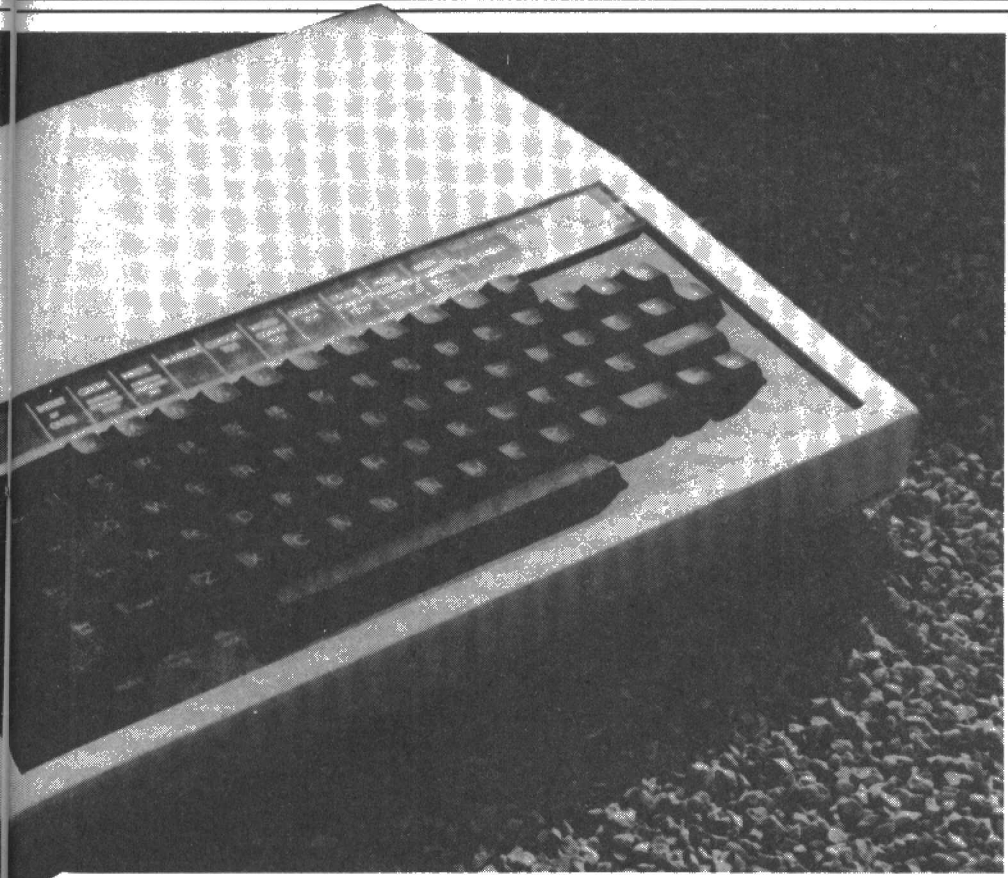
Figure 2 gives a list of all the devices at the various address locations, and shows the speed at which each is accessed. For those addresses which could be accessed at either speed, some indication is given as to how the access speed may be changed.

As suggested in **Figure 2,** a facility has been provided on the pcb for changing the access speed of certain devices. Even the sideways ROMs can be run at either 2 MHz or 1 MHz. This facility was provided because in the earliest machines the Basic ROM was put in the slot now occupied by the MOS ROM (IC 51) and the operating system was put in four 4K EPROMs working at 1 MHz in what are now the sideways ROM sockets. You could still slow down



CHIP SELECT LINES OF DEVICES REQUIRING 1MHz ACCESS
OR FUNCTION
REQUEST SIGNAL
2MHz
1MHz
÷ 2
SPEED-SWITCHING CIRCUITS
ϕ IN
6502A
2MHz
8MHz
DIVIDING CIRCUITS (PART OF VIDEO PROCESSOR CHIP)
16MHz CLOCK

Figure 1. Block diagram of the 1MHz/2MHz speed switching system.



the access of these ROMs to 1 MHz if you wanted to by using the selection links, but there seems little point in doing so.

The only device which has the specific option of being speeded up is the 1 MHz bus. Links S15 and S16 are provided in order to turn this into a 2 MHz bus! However it is also possible, by a simple hardware modification, to speed up the external VIA which provides the User Port and Printer Port facilities. Making this work at 2 MHz not only increases the interfacing speed of the ports but also improves the timing resolution of the VIA counter-timers from 1 to 0.5 microseconds. This will also enable us, amongst other things, to double the upper frequency limit of the frequency meter program that appeared in the September issue.

All the other devices that work at 1 MHz such as the serial processor and the analogue to digital converter are inherently slow devices anyway, thus it would be of no real advantage to up-rate them to 3 MHz. The extra 0.5 or 1 micro-second saved at each access would be insignificant.

The Timing

Executing a 6502 instruction like LDA &2000, ie using absolute addressing, takes a total of four clock cycles. It is made up of one cycle for fetching each of the three bytes of the instruction from the memory, and one cycle for actually executing this instruction and transferring the data from local &200 to the accumulator. Thus, using the 2 MHz clock, this would take 4 x 0.5 μS = 2 μS.

If instead, the address used was one of those which is accessed at 1 MHz, such as LDA &FE60, then the timing would be three cycles at 2 MHz. This is because the instructions, whether in RAM or ROM, are accessed at the higher speed, and then one cycle at the slower speed. In theory this should mean that it would take 3 x 0.5 + 1 x 1.0 μS = 2.5 μS. However, if you do some actual timings, you will discover that sometimes such an instruction takes 2.5 μS and sometimes it takes 3.0 μS.

LISTING 1. Test routine used to demonstrate the slow and fast accesses of the processor to 1MHz devices.

```
 10 CLS
 20 DIM CODE% 40
 30 REPEAT
 40    INPUT
    '"HEX ADDRESS",A$
 50    IF A$>"" THEN
address% = EVAL("&"+A$)
 60    P%=CODE%
 70    [OPT 2
 80    LDX #0
 90    LDY #0
100
110    .start
120    LDA address%
130    LDA address%
140    DEY
150    BNE start
160    DEX
170    BNE start
180    RTS
190    ]
200    TIME=0
210    CALL CODE%
220    X% = TIME
230 PRINT '"&";~address%;
" is being accessed at ";
240    IF X%<50 PRINT
"2 MHz" ELSE PRINT "1 MHz"
250    UNTIL0
260 END
```

This effect can be seen if you run **Listing 1** and use an oscilloscope to look at various waveforms within the computer.

These waveforms illustrated in **Figure 3** have, for simplicity, been drawn as if all changes of state occurred at exactly the

Address range	Device	Speed	Changed by
0000–7FFF	RAM	2 MHz	—
8000–BFFF	MOS ROM	2 MHz	—
C000–FBFF	IC 100	2 (1) MHz	S18
	IC 52	2 (1) MHz	S19 & D12
	IC 88	2 (1) MHz	S19 & D11
	IC 101	2 (1) MHz	S19 & D10
FCoo–FCFF	Fred	1 (2) MHz	S16
FD00–FDFF	Jim	1 (2) MHz	S15
FE00, FE01	6845 CRTC	1 MHz	—
FE08, FE09	6850 ACIA	1 MHz	—
FE10	IC 4 SERPROC	1 MHz	—
FE18	NMI intOFF/ Station ID	1 MHz	—
FE20	VIDPROC NMI intON	2 MHz	—
FE30	ROM select	2 MHz	—
FE40–FE4F	Internal VIA	1 MHz	—
FE60–FE6F	External VIA	1 (2) MHz	A simple hardware modification
FE80–FE84	FDC	2 MHz	—
FEA0–FEA3	ADLC (Econet)	2 MHz	—
FEC0–FEC3	A-D converter	1 MHz	—
FEE0–FEFF	Tube	2 MHz	—
FF00–FFFF	MOS ROM	2 MHz	—

Figure 2. A list of the access speeds of the various devices at different memory locations.

same instant. this is not quite true as there are delays of a few tens of nano-seconds. However, to represent them on a diagram of this scale, would make it even more confusing than it already is.

As you can see, the first access to a slow device (LDA &FC00) takes only 1.0 μS whereas the second (STA &FD00) takes 1.5 μS. The reason for this is that a full 1 MHz cycle (logic 0 then logic 1) is available as soon as the request signal appears the first time, whereas for the second access, the request comes when the 1 MHz signal is high. Therefore the system has to wait until the 1 MHz clock signal goes back to logic 0 before setting up the new address and transferring the data, ie it has to wait an extra 0.5 μS.

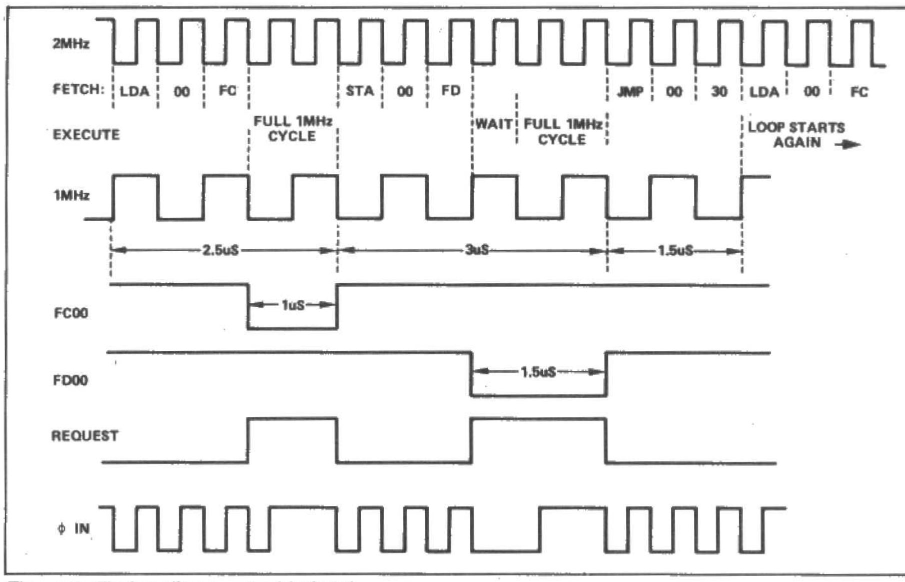


Figure 3. Timing diagram for Listing 1.

Speeding things up

To speed up the 1 MHz bus, the circuit diagram suggests that all you have to do is to cut links S15 and S16 which are actually made with track links on the pcb and then add pull-up resistors R72 and R73 (each 3k3), if they are not already fitted, at the places marked on the board, ie just along the right hand side of IC3. This should then increase the access speeds of pages &FD and &FC respectively (JIM and FRED).

Unfortunately there is an error on the pcb which has perpetuated right up to issue 7! There is nothing wrong with S15, but if you cut S16 and try to read memory locations &FC00 – &FCFF you will find that instead of looking at the I/O page you are looking at a normally inaccessible part of the MOS ROM!

"Unfortunately there is an error on the pcb which has perpetuated right up to issue 7!"

The reason for this is that pages &FC, &FD and &FE are actually within the memory address range of the operating system ROM and therefore when any of these addresses is accessed, the MOS ROM must be disabled. However, facility was provided by links S14 and S17 to re-enable the MIS ROM for pages &FC and &FD so that, during the development stages of the machine, the extra memory could be used as part of the operating system. Unfortunately due to a layout error on the pcb, breaking link S16 also disconnects the skip enable line from the gate which disables the MOS ROM so that the ROM is re-enabled. (As a matter of interest, what you would see is the beginning of a list of names which has been hidden, for posterity, in the three inaccessible pages of the MOS ROM. They are actually the names of some of the people involved in designing the BBC microcomputer!)

There is, however, no real difficulty in getting round this problem and changing the speed to 2 MHz. All you have to do is to break the track in between the North pad of S16 and pin 2 of IC23. This in fact, is not the inconvenience it might sound since, although S15 is made by a track on the upper side of the PCB, S16 is made by a track on the underside and breaking it would involve taking out the whole of the pcb. Breaking the track going to pin 2 can be done from above the board without moving the pcb at all.

If you need to be able to restore the speed of the bus to 1 MHz then you will have to solder a link directly from pin 2 of IC23 back to the North pad of S16. This, along with the contacts to S15, could perhaps be put onto a double pole switch which could be mounted on the back of the computer.

Speeding up the VIA

Although the facility is not actually provided on the pcb, it is also possible to increase the access speed of the external VIA to 2 MHz. This would mean replacing the 6522 (IC69) with a 6522A to ensure that it was capable of running at the full speed. Although some of the ordinary 1 MHz 6522's will still keep going at 2 MHz, if you are doing anything where loss of data would be serious, it is better to pay out for a 6522A.

LISTING 2. To test whether any given address location is being accessed at 1MHz to 2MHz. To repeat the test on the same address, just press return.

```
10 P%=&3000
20 [
30 SEI
40 .next
50 LDA &FC00
60 STA &FD00
70 JMP next
80 ]
90 CALL &3000
```

To implement this change, all you have to do is to break the track that crosses from pin 11 of IC23 and goes through in between pins 4 and 5 of IC24 and connects to pin 12 of that IC.

This increases the access speed, but the clock speed also needs to be increased. On the pcbs of issues 4 or 7 this is very simple because the 2 MHz line has been brought onto one of the chip select lines of the VIA (CS1 – pin 24) which happens to be immediately next to the clock line (pin 25). Thus, all you need to do is to bend out pin 25 of the 6522A so that when you insert it into the IC socket, this pin does not go into the socket where it would normally pick up the 1 MHz signal. Then, to link it to the 2 MHz line, a simple solder bridge between the shoulders of the two pins just next to the body of the chip is a crude but effective method!

On the pcbs of issue 3 or earlier, the CS1 line of the 6522 was tied up to +5V rather than having the 2 MHz signal applied to it. In this case you will have to bend out pin 25 of the VIA as before, but pick up the 2 MHz signal from pin 1 or pin 12 of IC25 (just north of IC23). This can be done with a few centimetres of insulated wire soldered onto the shoulder of say pin 1 of IC23 and pin 25 of IC69.

Having made this modification or the one for speeding up the 1 MHz bus, you can use **Listing 2** to check whether the access speed has in fact increased to 2 MHz.

Having looked this month at this technique of dual speed processing which the BBC micro uses, and seen how to speed up both the 1 MHz bus and the external VIA to 2 MHz, we will turn our attention next month to the software techniques necessary to make the most of this improvement in interfacing speed. In particular we will see how to generate high speed waveforms using an external digital to analogue converter.

Paul Beverley is the author of the Service Manual for the BBC microcomputer which will be on sale shortly.

DATAPAD

Jan Walker and J. Whetton explain how to set up a numeric keypad on the BBC Micro – a professional addition to anyone's machine.

One of the few features which the BBC micro lacks is a numeric data mode. This can make the entry of mathematical or financial data tedious. But for under ten pounds, you can attach a keypad to the user port, and the keys can be programmed to produce any ASCII code from 1 to 255.

Which keypad?

There are several types of keypad available with either 12 or 16 keys, wired linearly or in a matrix. Using a matrix-wired keypad allows direct connection to the user port without the need for additional circuitry. For this project a BT type 12-key pad arranged in a 4X3 matrix was chosen (see **Figure 1**) but a 16-key 4x4 matrix could

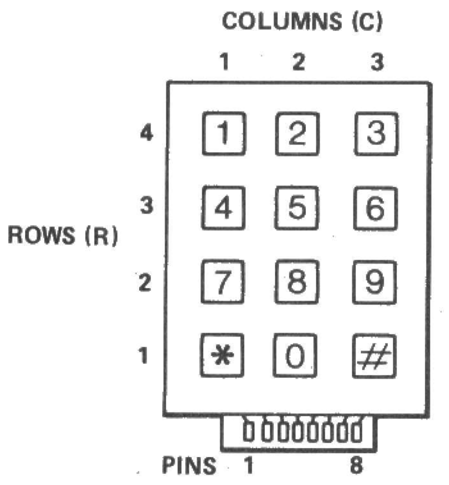


Figure 1. Keypad layout.

easily be used instead. Most keypads would need to be mounted in a suitable box.

Hardware

In a matrix keypad, pressing a key connects a column pin with a row pin (see

LISTING 1.

```
L    10PEM DATAPAD SOFTWAPE
   20REM VERSION 2.0
   30REM (C) 1984 WOODLEIGH SOFT
   40REM By Ivan Whetton and Ian Walker
   50REM Needs BASIC 2
   60REM
   70ddrb=&62 :REM data direction register of user port
   80orb=&FE60 :REM output register of user port
   90osbyte=&FFF4
  100IRQ1=&204 :REM primary interrupt vector
  110table1=&70:REM pointer (2 bytes) to keypress code table
  120table2=&72:REM pointer (2 bytes) to key ASCII code table
  130byte=&74
  140
  150MODE7
  160PROCtitle_page
  170PROCtype_of_use
  180ONERROR REPORT:z=INKEY(200)
  190PROCsave_code
  200ONERROR OFF
  210END
  220
  230
  240DEFPROCtitle_page
  250CLS
  260FORz=0TO1
  270PRINTTAB(10,z+2)CHR$(141)"DATAPAD PROGRAM"
  280NEXT
  290PRINT'''"1. Read instructions on use of datapad."
  300PRINT"2. Skip the instructions."
  310PRINT"3. Exit the program."''
  320PROCchoice(0,4)
  330IF N%=1 PROCinfo ELSE IF N%=2 THEN ENDPROC ELSE PROCexit
  340ENDPROC
  350
  360
  370DEFPROCinfo
  380REPEAT
  390CLS
  400PRINTTAB(6,3)"USING THE DATAPAD"
  410PRINTTAB(0,8)"1. Default values."
  420PRINT"   These allow the pad to be used as       a simple numeric keypad."
  430PRINT"   The  *  is used as a <DELETE> key."
  440PRINT"   The  £  is used as a <RETURN> key."
  450PRINTTAB(0,20)"Press  <SPACEBAR> to continue. "
  460z=GET
  470CLS
  480PRINT''"2. Defining values"
  490PRINT"     a) to produce a 12-key pad."
  500PRINT"        The keys are defined such that         each of the 12 keys pr
oduces a        particular ASCII code."
  510PRINT"     b) to produce a 16-key pad."
  520PRINT"        The * key and £ key are used as        shift keys."
  530PRINT"        Only the number keys in the same       column as the shift ke
y held          can be defined unambiguously.           eg. * + 7 , £ + 6  are
 valid.             * + 2 , £ + 4  are not."
  540PRINT"        The 10 number keys are defined         to produce a particula
r ASCII         code."
  550PRINTTAB(0,21)"Press  <SPACEBAR> to continue. "
  560PRINT"  or   <RETURN> to read again."
  570UNTIL GET<>13
  580ENDPROC
  590
  600
  610DEFPROCtype_of_use
  620CLS
  630PRINTTAB(10,2)"CONFIGURE KEYPAD"
  640PRINT''"1. Use the default values."
  650PRINT"2. Define ASCII values of keys."
  660PRINT"3. Exit the program."''
  670PROCchoice(0,4)
  680IF N%=1 THEN PROCdefault ELSE IF N%=2 PROCdefine ELSE PROCexit
  690ENDPROC
  700
  710
  720DEFPROCchoice(llimit,ulimit)
  730REPEAT
  740INPUT'"Enter the number of your choice. "N%
  750UNTILN%<ulimit AND N%>llimit
  760ENDPROC
  770
  780
  790DEFPROCdefault
  800RESTORE
  810data1$=FNread
  820data2$=FNread
  830ENDPROC
  840
  850REM Default valid keypress codes
  860DATA&3E,&3D,&3B,&5E,&5D,&5B,&6E,&6D,&6B,&76,&75,&73,0
  870
  880REM Default ASCII codes
  890DATA&31,&32,&33,&34,&35,&36,&37,&38,&39,&7F,&30,&D,0
  900
  910DEFFNread
  920C$=STRING$(13,"*")
  930C$=""
  940FORX%=0 TO 12
  950READA$
  960C$=C$+CHR$(EVAL(A$))
  970NEXT
  980=C$
  990
 1000
 1010DEFPROCassemble
 1020DIM code% 255
 1030FOR PASS =0 TO 1
 1040P%=load%
 1050O%=code%
 1060[OPT 4
 1070\ check vectors do not already pointto keypad code
 1080.set
 1090SEI
 1100LDA IRQ1
 1110CMP £(keys MOD 256)
 1120BNE change_vector1
 1130LDA IRQ1 +1
 1140CMP £(keys DIV 256)
```

KEY	C1	C2	C3	R4	R3	R2	R1	SC
1	●						●	
2		●					●	
3			●				●	
4	●					●		
5		●				●		
6			●			●		
7	●				●			
8		●			●			
9			●		●			
*	●			●				
0		●		●				
#			●	●				
PIN NO	1	2	3	4	5	6	7	8

Figure 2. Internal switch connections.

Figure 2). For example, pressing key 5 connects C2 with R2; pressing key 9 connects C3 with R3, and so on.

The 4x3 matrix gives 7 switch pins; pin 8 is a ground shield, unconnected to any of the switches. The switch pins 1-7 are connected via a 15-way ribbon cable to PB pins 0-6 on the user port; PB7 is tied to 0 volts together with keypad pin 8. The unused 0 volt lines are not connected (see **Figure 3**). The standard IDC header plug for the user port takes 20-way ribbon cable, so the 15-way cable must be placed at the left hand side so that keypad pin 1 is connected to user port pin 20. For a 16-key pad all PB lines would be used.

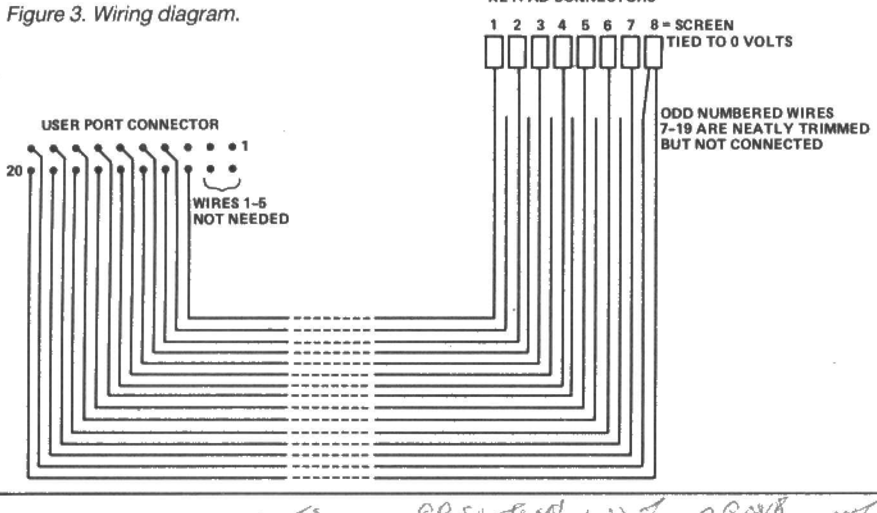
Decoding the keypad

The user port is set up so that the pins connected to the columns are outputs and the pins connected to the rows are inputs. Thus the output pins can be connected to the input pins by pressing the keys. For example, if pin C1 is written low (logic 0) then keys 1, 4, 7,* are able to pull pins R1, R2, R3, R4 low respectively. By reading the port we can see which, if any, of these pins has been pulled low, ie which of these keys has been pressed. Similarly, by writing to the other two output pins in turn other key presses can be detected. This combination of outputs and inputs produces a unique key number for each key (see **Table 1**).

```
1150BNE change_vector2
1160CLI
1170RTS
1180\ save old vectors and set IRQ1 to point to keypad code
1190.change_vector1
1200STA oldvec +1
1210LDA IRQ1 +1
1220.change_vector2
1230STA oldvec +2
1240LDA £(keys MOD 256)
1250STA IRQ1
1260LDA £(ke[illegible] DIV 256)
1270STA IR[illegible] +1
1280LDA £&96 \disable input latching
1290LDX £&6B
1300JSR osbyte \ACR bit 1 clear
1310TYA
1320AND £253 \ mask for bit 1
1330TAY
1340LDA £&97
1350JSR osbyte
1360LDX £&62 \PB0-2 outputs
1370LDY £&7  \PB3-7 inputs
1380JSR osbyte
1390LDA £(data1 MOD 256)
1400STA table1
1410LDA £(data1 DIV 256)
1420STA table1 +1
1430LDA £(data2 MOD 256)
1440STA table2
1450LDA £(data2 DIV 256)
1460STA table2 +1
1470CLI
1480RTS
1490\ restore vectors
1500.reset
1510SEI
1520LDA IRQ1
1530CMP £(keys MOD 256)
1540BNE rtn
1550LDA IRQ1 +1
1560CMP £(keys DIV 256)
1570BNE rtn
1580LDA £&93 \default for OS 1.20
1590STA IRQ1
1600LDA £&DC
1610STA IRQ1 +1
1620.rtn
1630CLI
1640RTS
1650\ main interrupt routine
1660.keys
1670PHP
1680PHA
1690TYA
1700PHA
1710TXA
1720PHA
1730BIT &FE4D \check interrupt is system timer1 interrupt
1740BVC out
1750JSR scan
1760CMP byte
1770BEQ out
1780STA byte
1790TAY
1800LDX £0
1810LDA £&8A
1820JSR osbyte
1830.out
1840PLA
1850TAX
1860PLA
1870TAY
1880PLA
1890PLP
1900.oldvec
1910JMP &FFFF \ dummy address
1920\ writes valid keypress values to user port until same value read back (tha
t key is pressed) or end of table reached.
1930.scan
1940LDY £&FF
1950.loop1
1960INY
1970LDA (table1),Y
1980BEQ end  \end of table marker
1990STA orb
2000LDA orb
2010CMP (table1),Y
2020BNE loop1
2030LDA (table2),Y \ read ASCII value for that key
2040.end
2050RTS
2060.data1
2070EQUS data1$  \ keypress codes
2080.data2
2090EQUS data2$  \ ASCII codes
2100]
2110NEXT
2120length%=O%-code%
2130ENDPROC
2140
2150
2160DEFPROCdefine
```

↓ See Below

Figure 3. Wiring diagram.


KEYPAD CONNECTORS
1 2 3 4 5 6 7 8 = SCREEN TIED TO 0 VOLTS
ODD NUMBERED WIRES 7-19 ARE NEATLY TRIMMED BUT NOT CONNECTED
USER PORT CONNECTOR
20
1
WIRES 1-5 NOT NEEDED

DO NOT ENTER LISTING UNTIL YOU HAVE READ THIS

WARNING

The following changes should be made to the listing printed above:

Delete line 170
Amend line 1040 P% = load %
Insert line 1045 O% = code %
Amend line 1050 [OPT 4
Amend line 2110 length % = 0% — code %
Delete line 3170
Amend line 3190 PRINT TAB (21,7) STRING$ (15, " ")
Amend line 3200 INPUT TAB (0,7) "Enter load address & "load $
Insert line 3225 PROCassemble
Amend line 3240 PRINT TAB (16,10) STRING$ (15," ")
Amend line 3250 INPUT TAB (0,10) "Enter file name "filename$

Note: This program requires Basic 2, but could be modified to run under Basic 1.

NOTE CORRECTION LIST ABOVE NOT REQUIRED - AS MAIN LISTING IS CORRECT - FROM E&CM JAN 1985 P 10 OF[illegible]

TABLE 1. Hex values for individual key presses.

	USER PORT PB LINES								
	TIED TO 0V	INPUTS				OUTPUTS			
	7	6	5	4	3	2	1	0	
KEY		R1	R2	R3	R4	C3	C2	C1	VALUE IN HEX
1	0	0	1	1	1	1	1	0	&3E
2	0	0	1	1	1	1	0	1	&3D
3	0	0	1	1	1	0	1	1	&3B
4	0	1	0	1	1	1	1	0	&5E
5	0	1	0	1	1	1	0	1	&5D
6	0	1	0	1	1	0	1	1	&5B
7	0	1	1	0	1	1	1	0	&6E
8	0	1	1	0	1	1	0	1	&6D
9	0	1	1	0	1	0	1	1	&6B
*	0	1	1	1	0	1	1	0	&76
0	0	1	1	1	0	1	0	1	&75
#	0	1	1	1	0	0	1	1	&73
PIN No. ON KEYPAD	8	7	6	5	4	3	2	1	

The software provided scans the keypad by writing the key numbers in turn to the user port and immediately reads back the value on the port. This value is the same as the key number if that key is pressed.

It is possible to expand the keypad beyond 12 keys by using one or more of the keys as a shift key. Unique key numbers are only produced when a shift key is used with other keys in the same column (see **Table 2**). The key defined as "shift" cannot now be used on its own.

TABLE 2. Hex values for two keys held together.

PB Lines

Key	7	6	5	4	3	2	1	0	Hex
*	0	1	1	1	0	1	1	0	&76
1	0	0	1	1	1	1	1	0	&3E
*-1	0	0	1	1	0	1	1	0	&36

The software

The software is in two parts – the Basic program **(Listing 1)** and the machine code which it generates.

The Basic program allows the user to choose the configuration of the pad and the ASCII codes produced by each key. Default values are provided. The program would only need slight modification if a 16 key pad were being used.

The machine code which is subsequently assembled can be saved and used on its own. A suitable place for the code in a disc-based system is the cassette/RS423 input buffer (&A00-&AFF). On system timer interrupt the keypad is scanned. If a key is pressed its ASCII code is obtained from a look-up table and put in the keyboard buffer.

Interesting features

Because each key on the keypad can be programmed to generate any ASCII code except zero, each can be made to behave exactly like a function key (eg to hold strings). If the break key (KEY10) holds a string, mimicking it on the keypad produces the string but not break.

Parts List

User port connector; 400mm 15-way (minimum) ribbon cable; Matrix keypad.

```
2170CLS
2180PRINTTAB(11,3)"DEFINE VALUES"
2190PRINT''"1. Define keypad as 12 keys."
2200PRINT"2. Define keypad as 16 keys."
2210PRINT"3. Exit the program."''
2220PROCchoice(0,4)
2230IF N%=1 PROCtwelve ELSE IF N%=2 PROCsixteen ELSE PROCexit
2240ENDPROC
2250
2260
2270DEFPROCtwelve
2280PROCtype
2290data1$=FNread
2300data2$=STRING$(13,"*")
2310data2$=""
2320twelve$="123456789*0£"
2330CLS
2340PRINTTAB(0,2)"LEGEND"TAB(13,2)"DEFINED AS "type$
2350FOR counter = 1 TO 12
2360PRINTTAB(3,counter+3)MID$(twelve$,counter,1)
2370REPEAT
2380REPEAT
2390PRINTTAB(16,counter+3)STRING$(12," ")
2400INPUT TAB(16,counter+3)input$
2410IF flag UNTIL FNletter ELSE UNTIL FNnumber
2420UNTILFNok(3)
2430data2$=data2$+input$
2440NEXT
2450IF FNproceed =FALSE GOTO2280
2460data2$=data2$+CHR$(0)
2470ENDPROC
2480
2490
2500DEFFNletter
2510IF LEN(input$)=1 =TRUE ELSE =FALSE
2520
2530
2540DEFFNnumber
2550LOCAL ok,bad
2560IFLEFT$(input$,1)="&" c$=RIGHT$(input$,LEN(input$)-1) ELSEc$=input$:GOTO263
0
2570FORi=1TO LEN(c$)
2580REM validates hex entries
2590IFASC(MID$(c$,i,1))<48OR(ASC(MID$(c$,i,1))>57AND ASC(MID$(c$,i,1))<65)OR AS
C(MID$(c$i,1))>70 bad=-1:i=LEN(c$)
2600NEXT
2610IFbad THEN=FALSE ELSEGOTO2640
2620REM validates decimal entries
2630IF VAL(c$)=0 =FALSE
2640IF EVAL(input$) <256 AND EVAL(input$) >0 ok = TRUE ELSE ok = FALSE
2650input$=CHR$(EVAL(input$))
2660=ok
2670
2680
2690DEFPROCtype
2700CLS
2710PRINTTAB(10,2)"KEY DEFINITION"
2720PRINT''"1. Define as a character."
2730PRINT"2. Define as an ASCII code."
2740PRINT"3. Exit the program."''
2750 PROCchoice(0,4)
2760IF N%=1 THEN type$="A CHARACTER.":flag=TRUE ELSE IF N%=2 THEN type$="AN ASC
II CODE.":flag=FALSE ELSE PROCexit
2770ENDPROC
2780
2790
2800DEFPROCsixteen
2810 data1$=CHR$(&3E)+CHR$(&3D)+CHR$(&3B)+CHR$(&5E)+CHR$(&5D)+CHR$(&5B)+CHR$(&6
E)+CHR$(&6D)+CHR$(&6B)+CHR$(&75)+CHR$(&36)+CHR$(&56)+CHR$(&66)+CHR$(&33)+CHR$(&5
3)+CHR$(&63)+CHR$(&0)
2820data2$=STRING$(17,"*")
2830data2$=""
2840sixteen$="1  2  3  4  5  6  7  8  9  0  *_1*_4*_7£_3£_6£_9"
2850PROCtype
2860CLS
2870PRINTTAB(0,2)"LEGEND"TAB(13,2)"DEFINED AS "type$
2880FOR counter=0 TO 15
2890PRINTTAB(2,counter+4)MID$(sixteen$,counter*3+1,3)
2900REPEAT
2910REPEAT
2920PRINTTAB(16,counter+4)STRING$(12," ")
2930INPUTTAB(16,counter+4)input$
2940IF flag UNTIL FNletter ELSE UNTIL FNnumber
2950UNTILFNok(4)
2960data2$=data2$+input$
2970NEXT
2980IF FNproceed = FALSE THEN 2850
2990data2$=data2$+CHR$(0)
3000ENDPROC
3010
3020
3030DEFFNok(offset)
3040PRINTTAB(24,counter+offset)"OK? Y/N";
3050z=GET
3060IFz=89 OR z=121 THEN PRINTTAB(24,counter+offset)STRING$(7," "):=TRUE ELSE P
RINTTAB(24,counter+offset)STRING$(7," "):=FALSE
3070
3080
3090DEFPROCsave_code
3100CLS
3110PRINTTAB(9,2)"SAVE MACHINE CODE"
3120PRINTTAB(0,5)"1. Save machine code."
3130PRINTTAB(0,6)"2. Exit the program."''
3140PROCchoice(0,3)
3150IF N%=2 PROCexit
3160CLS
3170PRINTTAB(9,2)"SAVE MACHINE CODE"
3180REPEAT
3190PRINTTAB(21,7)STRING$(15," ")
3200INPUTTAB(0,7)"Enter load address  &"load$
3210load%=EVAL("&"+load$)
3220UNTILload%>0 AND load%<&8000
3230PROCassemble
3240REPEAT
3250PRINTTAB(16,10)STRING$(15," ")
3260INPUTTAB(0,10)"Enter file name "filename$
3270UNTILLEN(filename$)>0 AND LEN(filename$)<8
3280IF FNproceed =FALSE THEN 3100
3290PRINT'"Saving machine code."
3300OSCLI "SAVE "+filename$+" "+STR$~(code%)+"+"+STR$~(length%)+" "+STR$~(load%
)+" "+STR$~(load%)
3310PRINT''"To enable keypad type  *"filename$"  (DISC)"
3320PRINTTAB(19)"or  */"filename$"  (TAPE)"
3330PRINT"To disable keypad type   CALL &";~load%+reset-load%
3340PRINT"To re-enable keypad type CALL &";~load%
3350ENDPROC
3360
3370
3380
3390
3400DEFFNproceed
3410PRINT'"Do you wish to proceed ? Y/N ";
3420z=GET
3430IF z=89 OR z=121 =TRUE ELSE =FALSE
3440DEFPROCexit
3450CLS
3460ONERROR OFF
3470END
>
```

COMMS COLUMN

Benedict Knox starts his new monthly communications column with a look at the Bulletin Board phenomena

There is little doubt that the computer communications revolution in Britain is coming to the boil. A number of factors have contributed to this: a slight liberalisation of British Telecom's policy toward connection of equipment to the telephone system; cheap (legal!) modems becoming readily available and the computer games market approaching saturation point.

Over the last year, I have used many 'online' systems, from Bulletin Boards in Britain to the huge information systems in the USA. In this regular column I will be reviewing most of the systems which are accessible over the telephone line, as well as giving hints, advice and news for those people who are already, or who would like to become involved with computer communications.

Over the last year, Bulletin Board Systems have become very popular amongst the computer communications fraternity.

One upon a time in America

Bulletin Boards originally started in America (what didn't?). A computer society at an American university wanted a more efficient method of distributing messages between its members. Up until then they had used the simple, but unreliable, method of posting messages on a particular noticeboard in the University foyer. The society developed a program to run on the University mainframe computer, which displayed messages and notices, on demand, to any of its members. The relatively unsophisticated program which resulted, was called 'The Electronic Bulletin Board'.

Since then, Bulletin Board Systems (BBS) have been further developed on microcomputers enabling people to run them from their homes. There are now over seven thousand Bulletin Boards in America, and about twenty here in Britain.

Why are they so popular? To answer that, a fuller description of a BBS is required.

A BBS is a free messaging system. The only requirement for membership of a BBS is that you have the right equipment to be able to access it (Computer, modem and software). Connection is then simply a matter of dialling the number of the particular BBS you are interested in (a list is given at the end of this article) and following the instructions as displayed on your screen.

Users may leave messages to each other in various sections of the BBS. If it is a private message, then it should be left in the 'Electronic Mail' section, where no-one other than the person named as recipient and the operator of the system (often called the 'Sysop', from System operator) can read it.

Most BBSs allow users to select specialised subject areas under which to send and receive messages. These 'Special Interest Groups' (SIGs) are set up by the Sysop and will – particularly on a new BBS – reflect his interests. Once the BBS has a larger base of regular users, then other SIGs are usually set up to cater for people with different makes of machine and/or other computing or non-computing interests: BBC, Sinclair, CP/M, communications, Sales and Wanted ads.

If you want to leave more general messages, or messages on subjects not covered by any of the SIGs, there is an 'Open' or 'General' section.

Operating systems

The messaging facilities described above are common to most BBSs. Which additional features are available will depend upon which system the BBS is running under. Three systems have become particularly popular in Britain: TBBS, CBBS and Forum 80.

TBBS, standing for The Bread Board System, is the easiest to use of the three, having extensive help facilities for new users, the level of which may be set by the user. Based upon menus, each option in a particular menu may be displayed with a full explanation, a single word, a single letter, or not displayed at all. To access an option, you need only type one of the letters which are clearly highlighted on your screen.

"Look out for the new multi-user bulletin board systems".

TBBS is named after the special type of circuit-board used by electronics enthusiasts (Bread Board), on which it is possible to set-up new circuits quickly and easily. The parallel is that on a TBBS system, extra SIGs and other sections, or

If you have any information on a new BBS or anything else which would interest other readers, please drop me a line, either at E&CM, or use Electronic Mail on TBBS London.

BULLETIN BOARD SERVICES

Name	Number	Notes (check times before calling)
BABBS	(0742) 667983	Times: 24hrs. British Apple System User Group
BBCBBS	(01) 579 2200	Times: 24hrs. BBC 'Micro Live'
CABB	(01) 631 3076	Times: 24hrs. Computer Answers magazine. 300 & 1200/75 baud.
CBBS London	(01) 399 2136	Times: Sun 1700-2200
CBBS Surrey	(04862) 25174	Times: 24hrs
CBBS SW	(0626) 890014	Times: 24hrs
City BBS	(01) 606 4194	Times: 24hrs 1200/75 Weds
Distel	(01) 679 1888	Times: 24hrs Commercial (free) BBS run by Display Electronics
Estelle (0279) 443511		Times: Office hours only. STC customers BBS
Forum-80 Hull	(0482) 859169	Times: Mon-Fri 1700-2330; 1200-2330 weekends
Forum-80 London	(01) 902 2546	Times: 1900-2200 wkdays; 1200-2200 wkends. Ring and ask for Forum-80
Hamnet	(0482) 497150	Times: 1800-0800. Radio Ham BBS.
Liverpool Mailbox	(051) 4288924	Times: 24hrs.
Mailbox-80 W. Midlands	(0384) 635336	Times: 1800-0800. Ring back
Manchester BBS	(061) 4273711	Times: Sun-Thu 2230-0000; Fri 2330-0200; Sat 2230-0200
Maptel	(0702) 552941	Times: 24hrs Maplin BBS Tele-ordering & stock levels
Microweb	(061) 4564157	Times: 24hrs Micro User magazine
NBBBS	(0827) 288810	Times: 24hrs Ring back
Southern BBS	(0243) 511077	Times: 2000-0200 Ring back
Stoke ITec	(0782) 265078	Times: 24hrs Remote CP/M
TBBS Blandford	(0258) 54494	Times: 24hrs
TBBS London	(01) 348 9400	Times 24hrs
TBBS Nottingham	(0602) 289783	Times: 24hrs Multi speed: 300/300, 1200/75 baud
TBBS Southampton	(0703) 437200	Times: 24hrs

'Boards', may be set-up with the minimum of effort.

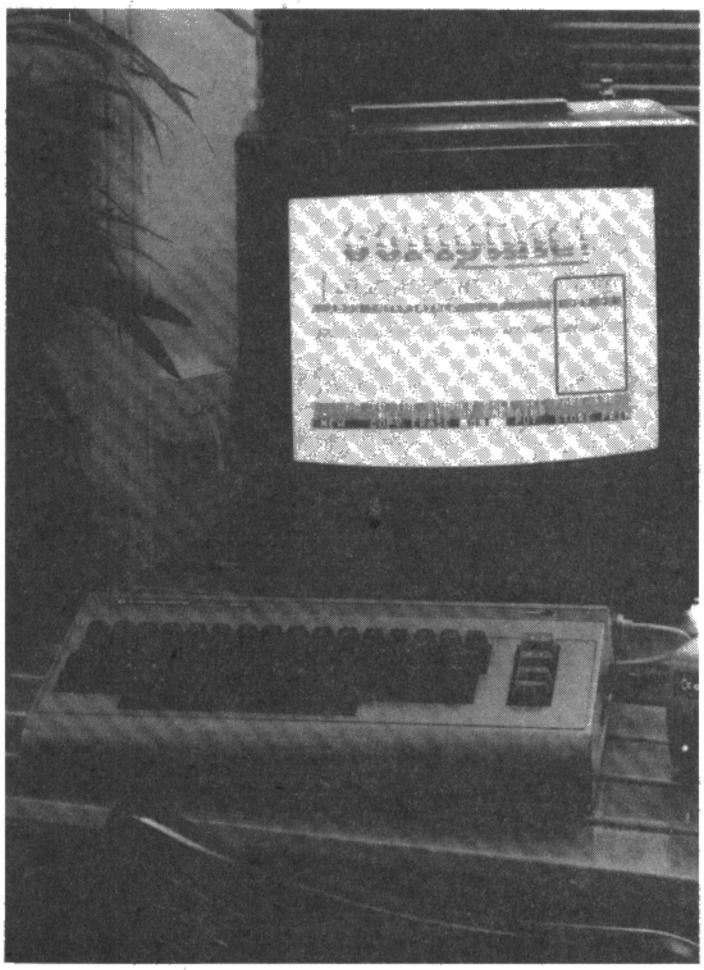
Compared to TBBS, CBBS (CP/M Bulletin Board System) and Forum 80 are distinctly unwieldy. In fact many commercial, supposedly user-friendly systems could learn a few things from TBBS.

CBBS and Forum 80 are very similar and also use the menu system, but appear to presume that most users are familiar with the systems, only giving command explanations when specifically asked for and then reversing through all the help levels described above: first a single letter 'aide-memoire', then a single word and finally the full explanation.

The AFPAS

To ensure that people may use the different systems with the greatest ease, the Association of Free Public Access Systems (AFPAS) was set-up. The main aim of AFPAS is to promote the standardisation of protocols among BBS'. To this end, the members have settled upon a 'word format' of: 7 bit word length, 1 stop bit, Even parity, or, in some circumstances, 8 bit word length, 1 stop bit, No parity. Additionally, the protocol to be used for error free file transferance, to exchange software, is the 'Christensen', XModem, CP/M, or Modem7 protocol (same protocol, different names).

Another function of AFPAS is to hold a blacklist of people who have for any reason, such as use of bad language, been banned from any of one its member BBSs. All other member Sysops have agreed to follow such bans on their own systems.



In the future we shall probably see more specialised BBSs appearing, as has happened in the USA. Another development to look out for is multi-user BBS, allowing more than one person access to the system at any one time. This, together with an increase in the number of BBSs will ensure that systems will no longer be engaged almost all day and night.

Guidelines for BBS use

1. Do not use false names or addresses.

2. always go through the logging off procedure when you want to leave a BBS. Do not just hang up.

3. Check operation times of a BBS before calling.

4. If a voice answers, do not hang up. The BBS may be off-line for some reason, or you may have the wrong number.

5. If a BBS uses a 'ring-back' system, use the following procedure: dial the BBS number, allow it to ring once or twice, hang up, call back again – the computer should answer this time.

A Hardware Guide For The BBC Microcomputer.

A 260 A4 PAGE BOOK WHICH INCLUDES:-

★ Comprehensive circuit description.
★ Full step-by-step details on upgrading Model 'A' to Model 'B'.
★ Full details on fitting disc and speech expansions.
★ Full explanation of all link settings and functions.
★ Full of circuits, hints, tips and modifications (many previously unpublished).
★ Full manufacturers data sheets on all major IC's including:- R6522, 8271, 6502, MC6845, MC6850, SN6489AN, UPD7002, SAA5050, HM4816, 2764, 27128, 7400 series pin-outs.

ALL DEALER ENQUIRIES WELCOME

A MUST FOR BOTH ENTHUSIAST AND ADVANCED USER ALIKE

To order send cheque/PO for £11.95 plus £1.50 p&p (UK only)
(Overseas readers — printed paper rate surface mail — £3.00 p&p — air mail extra).

ORDER FORM

Please send ....copies of "A Hardware Guide for The BBC Microcomputer"

At £11.95 each plus £1.50 p&p (UK). I enclose cheque/postal order for £................

NAME ....

ADDRESS ....

....

All orders and enquiries to
WISE-OWL,
Hull Innovation Centre, Guildhall Road, Queens Gardens, Hull HU1 1HJ.

Multitasking made easy

Adam Denning says there's nothing to QL multitasking – he explains in his crash course on getting the most out of the QL.

Multi-tasking on the QL is easy. There really is nothing to it. All the rumours about how difficult it is to write programs which will run concurrently with others are pure fabrication. Oh sure, you have to be able to program in 68008 machine code, but let's face it if you're reading this magazine then it's a fair bet that you're half way there already.

Each multi-tasking program is referred to as a **job** by QDOS, and each is simply a self-contained program. Each job has its own record of all its registers, so whenever the system switches the processing from one job to another all it has to do is save the registers in a pre-defined place, load the registers of the next job, and away it goes.

Each job runs for the most part in the 68008's User mode, so that the task of processing each job can be looked after by the scheduler, which runs in Supervisor mode. If at any time a job wanted to go into Supervisor mode then it is fully able to do so, but the act of entering this mode immediately makes the current job the only task running until the system reverts to user mode.

The SuperBasic interpreter is itself a job, but is a little special in that it can alter the amount of room it takes up dynamically, while other jobs are required to declare their RAM requirements before they are activated. The SuperBasic interpreter is known as job 0, as it has a job number of 0 and a tag of zero. Don't worry, these words will be explained very soon.

"Oh sure, you have to be able to program in 68008 machine code".

Each job has a priority, which is a number between 0 and 127 which determines how often the job will be executed in a given period of time. QDOS multi-tasks by allocating each active job a slice of the processor's time, and it decides how often to give a job a time slice from that job's priority. If the priority is zero then the job will never be executed, it is said to be inactive. Other levels of priority mean that job is active and will therefore be given processor time in due course.

However a job can be active but in a state of suspension, either because of some deliberate act or because it is waiting for some action to finish. It may for

LISTING 1. The header file – header ASM.

```
         NOLIST

* Operating system vectors

UT_CON    EQU       $C6
UT_SCR    EQU       $C8
UT_ERRO   EQU       $CA
UT_ERR    EQU       $CC
UT_MINT   EQU       $CE
UT_MTEXT  EQU       $D0
UT_CSTR   EQU       $E6
CN_DATE   EQU       $EC
CN_DAY    EQU       $EE
BP_INIT   EQU       $110
CA_GTINT  EQU       $112
CA_GTFP   EQU       $114
CA_GTSTR  EQU       $116
CA_GTLIN  EQU       $118
BV_CHRIX  EQU       $11A
RI_EXEC   EQU       $11C


* Operating system offsets and equates

CH_LENCH  EQU       $28
BV_CHBAS  EQU       $30
BV_RIP    EQU       $58
ERR_NO    EQU       -6
ERR_EF    EQU       -10
ERR_BP    EQU       -15
ERR_OV    EQU       -18
ERR_BL    EQU       -21
RET_STR   EQU       1
RET_FP    EQU       2
RET_INT   EQU       3
OPEN_INX  EQU       0
OPEN_INS  EQU       1
OPEN_NEW  EQU       2
OPEN_OVR  EQU       3
OPEN_DIR  EQU       4

* Trap keys
* Trap 1

MT_CJOB   EQU       1
MT_JINF   EQU       2
MT_FRJOB  EQU       5
MT_SUSJB  EQU       8
MT_RELJB  EQU       9
MT_ACTIV  EQU       $A
MT_PRIOR  EQU       $B
MT_DMODE  EQU       $10
MT_IPCOM  EQU       $11
MT_RCLCK  EQU       $13
MT_ALCHP  EQU       $18
MT_RECHP  EQU       $19
* Trap 2

IO_OPEN   EQU       1
IO_CLOSE  EQU       2

* Trap 3

IO_FBYTE  EQU       1
IO_FLINE  EQU       2
IO_FSTRG  EQU       3
IO_SBYTE  EQU       5
IO_SSTRG  EQU       7
SD_WDEF   EQU       $D
SD_CURS   EQU       $F
SD_POS    EQU       $10
SD_TAB    EQU       $11
SD_NCOL   EQU       $14
SD_CLEAR  EQU       $20
SD_FOUNT  EQU       $25
FS_POSAB  EQU       $42
FS_POSRE  EQU       $43
FS_HEADR  EQU       $47
FS_LOAD   EQU       $48

* RI operation keys

RI_FLOAT  EQU       8
RI_ADD    EQU       $A
RI_MULT   EQU       $E
```

instance be waiting for another, subsidiary job to terminate, or it may want to read the keyboard but is unable to because another job is doing so.

In the November issue of *E&CM* I explained that most of the calls to the QDOS operating system are made by using one of five 68008 trap instructions: TRAP #0 to TRAP #4. Each of the individual traps is delegated a certain group of tasks, and in the case of traps with more than one function to provide (TRAPs #1, #2 and #3) the actual function is determined by the value of register D0 on entry to the trap. A trap is just a 68000 instruction which switches the processor to an exception handling routine, which is essentially similar to the software interrupt found on smaller processors like the 6502 and 6809. It also invokes supervisor mode, so anything that is trap invoked is likely to be executed within that job's current time slice. In certain cases this is guaranteed and is then described as an atomic action.

TRAP #1 is used by QDOS for all its manager routines, such as allocating memory to jobs, creating jobs, talking to the 8049 co-processor and so on. TRAP #2 is used to allocate RAM for I/O (in other words it looks after channel open and close) and TRAP #3 looks after other I/O operations such as the actual transfer of data, the screen drivers and the file system.

A job is normally held in a file, and would be executed from SuperBasic using either the EXEC or EXEC_W commands. The former loads and activates the job and then returns to the Basic interpreter, whilst the latter loads and executes the job but suspends the SuperBasic interpreter pending completion of the thus loaded job. Let's look at EXEC and EXEC_W in more detail, so that we can get a little more insight into QDOS and job control.

Open given filename for reading
Load the file's header into RAM
Read its length and the length of its data area
Create a job with these specifications
Load the file into the space allocated
Close the file
Activate the job with priority 32, and a timeout of 0 if the command was EXEC or —1 if the command was EXEC_W

That't the basic outline of how these two SuperBasic commands are implemented. Now let's examine each stage in greater detail. Remember that both these keywords are passed a filename as their parameter. We'll call this file job file. Having invoked the procedure it first checks to see if it has the correct number of parameters and whether they (or rather, it) is in the cor-

LISTING 2. Printer spooler routine.

```
* By Adam Denning
* Copyright (C) 1984 Adam Denning

         GET       "mdv1_header_asm"
         SIZE      150

* A printer spooler routine
* started 23rd September 1984
* Altered from byte-by-byte transfer to 100 bytes at a time
* on Wednesday 26th September
* Altered from 100 byte at a time transfer to 4096 bytes at a time
* transfer on 10th October 1984

BUF_LEN  EQU       100                 Length of input buffer
HEAP_ROOM EQU      4096                Length of common heap area wanted

         BRA.S     START_P             Ignore standard format code
         DC.L      0
         DC.W      $4AFB               Standard format identification
         DC.W      7                   Program name
         DC.B      'SPOOL_1',0

* Use MT_PRIOR to set the priority of this job to 1 so that it does not
* slow Basic down too noticeably. MT_PRIOR sets the priority of the job
* whose ID is held in D1 to the value held in D2. It preserves all
* registers except A0 which is left holding the base of this job's job
* control area. If D1 was passed as -1 (for current job) then on return
* it will hold the true job ID

START_P  MOVEQ     #MT_PRIOR,D0        Set priority
         MOVEQ     #-1,D1              of this job
         MOVEQ     #1,D2               to 1
         TRAP      #1

* Use the UT_CON utility routine to open a console device (CON_) with the
* parameters specified in the definition block pointed to by A1. It returns
* with the ID of the newly opened channel in A0 and corrupts D0 - D3 and
* A1 - A3. If there are no errors then D0 is returned as 0

         LEA.L     PBLOCK,A1           Open up specified console device
         MOVE.W    UT_CON,A2
         JSR       (A2)
         TST.L     D0                  Error?
         BNE       JOB_END             Yes, so leave

* Use the UT_MTEXT utility routine to print a prompt on the screen. This
* utility needs the ID of the channel to which the message is to be
* printed in A0 and the base address of the string in A1. It corrupts all
* registers except A0 (A4 - A7 are safe)

GET_FILE  LEA.L    MESSAGE1,A1         Print 1st message
         MOVE.W    UT_MTEXT,A2
         JSR       (A2)

* Use the IO FLINE trap to collect a filename from the keyboard (the
* input device to our console channel). IO_FLINE requires the ID of the
* channel from which the data is to be read in A0, a timeout in D3, the
* length of the buffer to which the line will be collected in D2 and
* the actual address of the buffer in A1. All registers are preserved
* except D1 which holds the number of bytes collected (including the L/F
* which terminated the input) and A1 which is updated to the next free
* buffer position. The error 'buffer full' can be returned but it is
* unlikely to be in this case; we ignore it anyway!

         MOVEQ     #BUF_LEN,D2         Fetch filename from channel
         MOVEQ     #-1,D3
         LEA.L     BUFFER,A1
         MOVEQ     #IO_FLINE,D0
         TRAP      #3
         MOVE.L    A0,-(A7)            Save console channel ID

* Now open the collected filename for shared input (so that other jobs
* can examine the file if necessary) using IO_OPEN. This trap requires
* the job ID in D1 (or -1 for current job) and the open access in D3 as
* a byte key. A0 must point to the name of the device to be opened. All
* registers are preserved except D0 (error return), D1 (true job ID if -1
* was passed) and A0, which holds the ID of the newly opened channel

         LEA.L     BUF_POS,A0          Get ready for IO_OPEN call by
         SUBQ.L    #1,D1               converting line fetched to a
         MOVE.W    D1,(A0)             string, removing LF from count
         MOVEQ     #OPEN_INS,D3        Open this file for input
         MOVEQ     #-1,D1
         MOVEQ     #IO_OPEN,D0
         TRAP      #2
         TST.L     D0                  Error?
         BEQ.S     GOT_FILE            No - so continue

* If the call to IO_OPEN above resulted in an error then the error code
* is in D0. The UT_ERR utility routine will print out the error message
* corresponding to this code to the channel whose ID is held in A0. All
* registers are preserved by the call

         MOVE.L    (A7)+,A0            Else retreive console channel ID
         MOVE.W    UT_ERR,A2           and write requisite error message
         JSR       (A2)                to it. Then try again.
         BRA.S     GET_FILF

* Console channel ID is currently on the stack; swap with file ID held in
* A), putting the console ID in A0 so that IO calls use that channel

GOT_FILE  MOVE.L   (A7)+,A1            Put console channel ID in A1
```

rect format.

It then uses trap #2 to open the file. The actual function performed is IO_OPEN, which involves D0 holding the code for IO OPEN (1), D1 holding the ID of the job for which the file is being opened (the SuperBasic interpreter – job 0 – in this case), D3 holding the code under which the file will be opened (probably 1 for shared read only) and A0 pointing to the filename. The trap is then executed and the channel is opened. Or not. If there was some error, such as the file not being found or not enough memory, the D0 is returned holding some value other than 0. This applies to all QDOS traps, D0 being equal to zero is the only indication of success. Any other values of D0 are actual error codes, which would normally be passed back to Basic and reported.

Assuming success, all other registers except A0 are returned unharmed, but A0 holds what is known as the channel ID. This, like the job ID is a long word (four bytes), and is of fundamental importance. To communicate with a channel you must have its ID. Note that there is no direct connection between the #channel numbers used in Basic and the ID returned by IO_OPEN.

Having successfully opened our file we must now read its header into RAM. A header is in this case 14 bytes long (assuming a microdrive file) and is laid out as shown:

long word	file length
byte	access (unused)
byte	type (0 for all except SEXECd files, when it is 1)
long word	data space length for SEXECd files or zero for normal files
four bytes	currently unused

All EXEC / EXEC_W needs to do is check whether or not the file type byte is 1 and if it is read the file length into register D2 and the data space length into D3. The header would be read into RAM with the FS_HEADR trap, which is trap #3 with D0 = #47. This requires D2.W to hold the buffer length (minimally 14, but 15 for safety), D3.W to hold the timeout (see below), A0 to hold the channel ID and A1 to point to the address of the buffer into which the header will be read. Assuming no errors the file length and data lengths can then be read with two simple MOVE.L instructions, and away we go.

The next trick is to create the job. The manager trap MT_CJOB does this, and it is invoked by loading D0 with 1 and doing a trap #1. It requires D1 to hold the ID of the job which will own the new job, and as this is going to be the SuperBasic interpreter this would normally be zero. D2 and D3 hold the code length and data lengths respectively and A1 holds an absolute starting address or zero. This would normally be zero as we don't want to rely on absolute addresses in a 68000 system. Putting 0 in A1 makes QDOS allocate

LISTING 2 – Continued

```
         MOVE.L    A0,-(A7)              save file channel ID on stacl
         MOVE.L    A1,A0                 make console current channel

* Use UT_MTEXT and IO_FLINE as before to collect the device name of the
* channel to which the data is to be printed

GET_OUTP LEA.L     MESSAGE2,A1           Print 2nd message to console
         MOVE.W    UT_MTEXT,A2           to it.
         JSR       (A2)
         MOVEQ     #BUF_LEN,D2           Get printer device specification
         MOVEQ     #-1,D3                from console and open it as file
         LEA.L     BUFFER,A1
         MOVEQ     #IO_FLINE,D0
         TRAP      #3
         MOVE.L    A0,-(A7)              Save console channel ID
         LEA.L     BUF_POS,A0            Point to start of string
         SUBQ.L    #1,D1                 'Remove' trailing L/F by reducing
*                                        the character count by 1
         MOVE.W    D1,(A0)               Save count at start of string

* Use IO_OPEN again to open the printer device for exclusive output

         MOVEQ     #OPEN_NEW,D3
         MOVEQ     #-1,D1
         MOVEQ     #IO_OPEN,D0
         TRAP      #2
         TST.L     D0                    Error?
         BEQ.S     GOT_OUTP              No - so continue

* If IO_OPEN above failed then use UT_ERR as before to print error message
* to console channel; then collect name again and repeat until no error

         MOVE.L    (A7)+,A0              Retrieve console channel ID
         MOVE.W    UT_ERR,A2             print requisite error message
         JSR       (A2)
         BRA.S     GET_OUTP              and try again

* We no longer need the console so we can cloose that channel. First
* swap its channel ID at the top of the stack with that of the newly
* opened printer device and then call IO_CLOSE. This trap just needs the
* channel ID in A0 and all registers except A0 are preserved

GOT_OUTP MOVE.L    (A7)+,A1              Swap console and printer channel
         MOVE.L    A0,-(A7)              IDs on stack and close the
         MOVE.L    A1,A0                 console device.
         MOVEQ     #IO_CLOSE,D0
         TRAP      #2

* Now allocate some common heap space for a large buffer to transfer
* bytes from the file to the printer. We first try to claim as many bytes
```

```
* as we can (HEAP_ROOM), bbut if this fails then D0 will contain an
* error code. If this is so then we divide the amount of room we require
* by 2 and repeat the process until we have space or until D1 is less
* than or equal to 1, when we return with no transfer as there is not
* enough room. MT_ALCHP requires the job ID in D2 (or -1 for the current
* job) and the number of bytes required in D1. All of D0 to D3 and A0 to
* A3 are corrupted, with D1 holding the number of bytes allocated and A0
* pointing to the beginning of the area allocated

         MOVE.L    #HEAP_ROOM,D1
GET_ROOM MOVEQ     #-1,D2
         MOVEQ     #MT_ALCHP,D0
         TRAP      #1
         TST.L     D0                    Error?
         BEQ.S     GOT_AREA              No

         LSR.L     #1,D1                 Divide space required by 2
         CMPI.L    #1,D1                 If D1<= 1 then stop with error
         BGT.S     GET_ROOM              else try again
         BRA.S     FILE_END

* Once area is allocated save address of area in A3 and bytes in area in
* A2, as neither of these registers are corrupted by later traps

GOT_AREA MOVE.L    D1,A2
         MOVE.L    A0,A3

* Collect input file ID from stack and use IO_FSRTG to read bytes from it
* into common heap buffer. IO_FSTRG fetches a string of bytes from the
* channel whose ID is in A0. It will fetch up to D2.W bytes depending on
* the timeout in D3.W and EOF being reached. The base address of the
* buffer must be in A1. It returns with the number of bytes collected in
* D1.W, an error code or zero in D0, and A1 updated to point to the next
* free position in the buffer. All other registers are preserved

FILE_P   MOVE.L    4(A7),A0              Get file channel ID from stack
         MOVEQ     #IO_FSTRG,D0          Fetch <HEAP_ROOM bytes from file
         MOVE.L    A2,D2                 Put area length in D2
         MOVEA.L   A3,A1                 and area start in A1
         MOVEQ     #-1,D3                infinite timeout
         TRAP      #3
         MOVE.L    D0,-(A7)              Save error return

* Use IO_SSTRG to send these bytes to the printer device. This trap
* requires the ID of the channel to which the bytes are to be sent in A0,
* the timeout in D3 (it's preserved from IO_FSTRG, so we don't need to
* set it again) and the base of the buffer from which the bytes are to be
* fetched in A1. It returns with an error report or zero in D0 (we ignore
* it anyway), the number of bytes sent in D1.W and the updated buffer
* position in A1. All other registers are preserved.
```

LISTING 2 – Continued

```
          MOVEQ    #IO_SSTRG,D0         Send these bytes to printer
          MOVE.L   4(A7),A0             Retrieve printer channel ID
          MOVEA.L  A3,A1                put area address in A1
          MOVE.L   D1,D2                and bytes collected in D2
          TRAP     #3

* If IO_FSTRG returned EOF then the process is over, so we can leave

          MOVE.L   (A7)+,D0             Retrieve error return
          TST.L    D0                   End of file?
          BEQ.S    FILE_P               No, so continue

* Use IO_CLOSE to close the printer channel then the file channel, both
* of who's IDs are on the stack.

FILE_END  MOVE.L   (A7)+,A0             Yes, so close printer
          MOVEQ    #IO_CLOSE,D0
          TRAP     #2
          MOVE.L   (A7)+,A0             and close file
          MOVEQ    #IO_CLOSE,D0
          TRAP     #2
          BRA.S    END_JOB

* Now force release the job (a bit heavy, perhaps, but it certainly
* ensures success!). Also, if one of the earlier errors causes a jump to
* this point then we can use UT_ERRO to send the error message to the
* command channel. UT_ERRO is identical to UT_ERR except that it requires
* no channel ID as it automatically uses channel 0 (if it's free, channel
* 1 if not).
```

```
JOB_END   MOVE.W   UT_ERRO,A2
          JSR      (A2)

END_JOB   MOVEQ    #MT_FRJOB,D0         Then kill this job.
          MOVEQ    #-1,D1
          TRAP     #1

* Console device specification

PBLOCK    DC.W     0                    No border
          DC.W     4                    black paper green ink
          DC.W     440                  width
          DC.W     30                   height
          DC.W     36                   X position
          DC.W     15                   Y position

MESSAGE1  DC.W     12
          DC.B     'Print file:


MESSAGE2  DC.W     16
          DC.B     'Printer device:


BUF_POS   DC.W     0

BUFFER    EQU      *

          END
```

an area for the job, and this will be its starting address. The trap returns with the ID of the new job in D1 and the base address of the area allocated in A0.

Now we need to load the code of the job from **job_file** into the area allocated. We use another trap #3 routine here, FS_LOAD. This is DO = $48, with D2 containing the length of the file, D3 containing the timeout, A0 holding the channel ID and A1 holding the address to which the file will be loaded.

We now call IO_CLOSE (trap #2, D0 = 2) to close the file whose channel ID is in A0 when we make the trap, and all we need to do now is activate the job. This is done with another trap #1 routine, MT_ACTIV (D0 = $A), which requires the ID of the job to be activated in D1, the priority with which it is to be activated in D1, the priority with which it is to be activated in D2 and the timeout in D3. Here the timeout is zero if the command invoking all this was EXEC, or –1 if we used EXEC_W.

That's it, really, but there's little point in us writing a program to do all this when all we need to do is type EXEC filename, is there? Instead we're going to explain job ID2, channel IDs and timeouts a little more and then write a program which will multi-task.

IDs and timeouts

To recap, whenever a channel is opened or job created, QDOS returns a number which uniquely identifies that job or channel to the system. This is of course its ID, which in both cases is a long word of data. This long word can be further split up into two words, the first of which bears a relationship to the number of channels opened or jobs created since the machine was last reset, and the lower word of which has something to do with the actual number of jobs or channels currently present.

The SuperBasic interpreter has an ID of zero, and we can never change this – we can't remove the interpreter from the job list and then recreate it later to give it another job ID. The next job to be created will be the given the ID $00000001, which corresponds to a job number of 1 and a 'tag' of zero. If we created another job we would find that its ID is $00010002, which means that it is job number two with a tag of 1. If we then killed our first job and loaded in another, this last job would be given a job ID of $00020001, as the tag still rises but the job number is allocated according to the number of jobs in the machine at the moment.

Later on we'll find that it is often useful to talk of a job in terms of its job number and its tag, as these two numbers individually are a lot easier to remember than the entire long word ID – especially if we choose to represent it in the rather more natural decimal.

An awful lot of the QDOS routines require a timeout. This is simply a measure of the time that the processor will spend doing the job of the particular trap before it returns to the calling program with failure or partial success. If the timeout is zero then the routine will, in most cases, try to do what it can. For example if we called a trap to collect bytes from a channel with zero timeout then it would collect as many bytes as there are to get. If we called the routine with infinite timeout (ie –1) then the trap would not return until it has either finished completely or had an error condition. The only real exception to this is MT_ACTIV, which has only two valid timeout values – 0 and –1. A timeout of zero in this case activates a job and resumes, while a timeout of –1 activates a job and waits for it to finish. The upshot of this is that the two most common timeouts that we are likely to use are 0 and –1. Naturally most of the routines which accept timeouts become non-atomic if the timeout is not zero.

The printer spooler

Now for our program. This is a very simple printer spooler routine which allows you to print a document (or indeed copy a file to any other device or file) while other operations

"The routine allows you to print a document as a background or foreground task".

such as a Basic program are going on. By setting the priority of this job to 1 we ensure that it is the most background of background tasks and is therefore unable to alter the speed of the SuperBasic interpreter very much. If on the other hand we wanted the print to continue as the foreground task whilst Basic or whatever proceeded as something less important, all we need to do is change the priority to some other value. Remember that until we do something about it the SuperBasic interpreter and all jobs activated by it have a priority of 32.

For the purposes of this article we must assume that you have an assembler, and in fact if you have the Metacomco assembler then you will not need to change any of the source. Other assemblers follow slightly different conventions, particularly in the area of assembler directives, so check your assembler documentation carefully.

The file mdv1_header_asm **(Listing 1)** is first included in the assembly. This file just consists of all the QDOS declarations, keys and equates which we use later on. We then declare the data size of the program as being 150 bytes. If your assembler doesn't have a directive like this, the best thing to do is to load the assembled code into a safe area of RAM and then resave it using SEXEC.

The program itself starts at the BUF_LEN declaration. The symbol BUF_LEN is 100 and the symbol HEAP_ROOM is 4096; we'll be using these a little later. The first instruction in the program is a BRA which jumps over a few bytes which declare the job as being in standard QDOS format (ie the word starting at the 6th byte is $4AFB and this is followed by the job name).

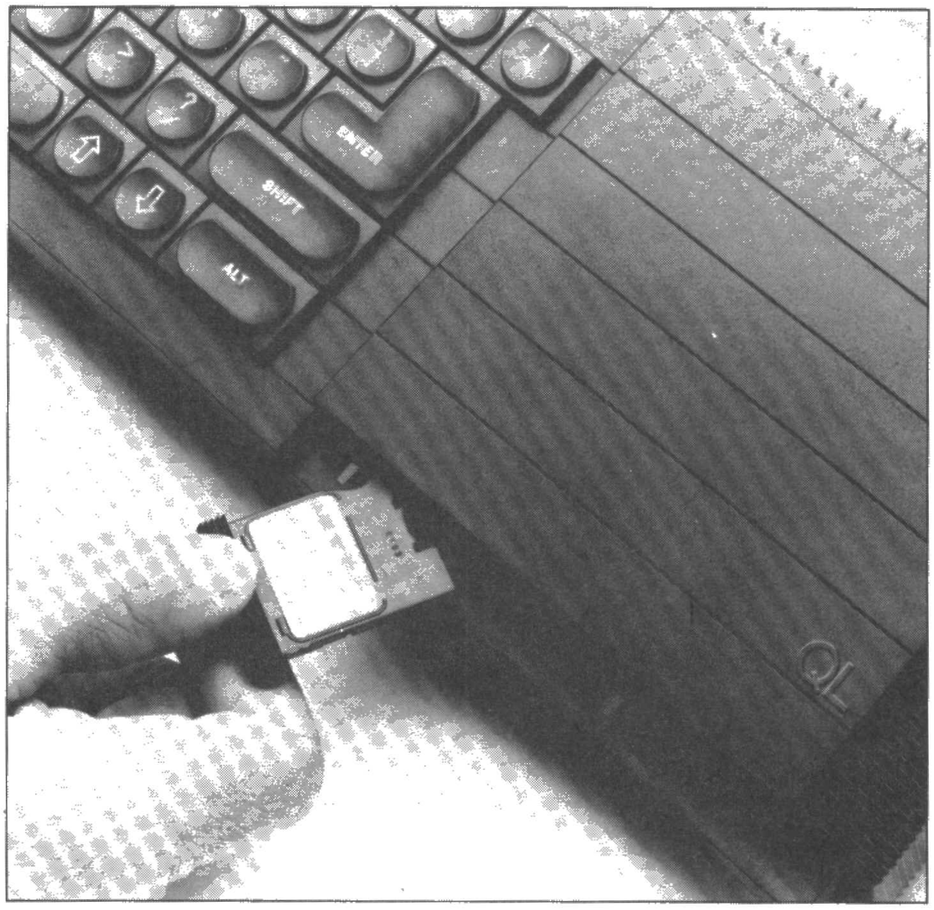
The main code starts at START_P. The MT_PRIOR trap is used to set the priority of the current job (indicated by a job ID of –1) to 1. We then use one of the thoroughly useful utility routines to open a console device. The various parameters of the window of this device are declared in a parameter block further down in memory called PBLOCK. UT_CON calls IO_OPEN and returns the ID of the newly opened channel in A0. If there was an error then D0 is non-zero, so we test for this. Assuming everything is OK we then print a message on our console which says 'Print file: '. This is a message to the user telling him to type in the name of the file which he wishes to print. To collect the filename we use the IO_FLINE trap, which is very useful as it allows the line to be edited before ENTER is pressed, in just the same way as in a line of Basic. The program's data space starts at BUFFER, to collect the filename.

"At any one time I've had up to 61 jobs running on a machine simultaneously".

When IO_FLINE returns D1 contains the number of characters collected, including the line feed which terminated the input. We're going to use the filename in a call to IO_OPEN, so we first have to get the string into QDOS format. A QDOS string is a string of characters preceded by a word containing the character count; a word is reserved at BUF_POS where we can put the string length. The first thing to do is decrease the length of the string by 1, as we don't want to send the line feed to IO_OPEN. We now store this string length in BUF_POS and call IO_OPEN to open the file. Notice that D3 contains the code for open or shared read, so that other programs can examine (but not write to) the file at the same time as this one.

If all succeeds here then save all the channel IDs on the stack and print another message, 'Printer device: '. This is a prompt for the channel on which we have the printer. We could enter ser1c, as I do for my Epson, or mdv1_dump to save the text to a file called mdv1_dump. Any valid QL device which supports output can be entered in response to this prompt. We again use IO_FLINE to read in the new string, taking the same precautions as before. Once a valid filename is obtained it is opened for exclusive output and the console closed as it is no longer needed.

The next bit is rather flash. The area between the system variables and the transient program area is called the common heap space, and is available for use by any job that



needs space. So we ask for 4096 bytes of it. If we don't get it, we divide the amount we're asking for by 2 until eventually we either get enough room or discover that there is absolutely none available. In the latter instance we just kill the job and return to Basic. If we can get the room then we enter a loop in which data is read from the input file using IO_FSTRG and sent to the output file using IO_SSTRG. IO_FSTRG is very similar to IO_FLINE except that we tell it how many bytes we want to collect, and a line feed is not taken as a terminator. IO_SSTRG is exactly its converse – it sends a predetermined number of bytes to a channel. When IO_FSTRG has reached the end of the input file, it returns End Of File, but nothing is done about that until AFTER the IO_SSTRG call as there may be some residual bytes to send.

When all is well and the end of file has been reached, we close our two files and kill the job. Killing a job requires a call to MT_FRJOB ('force release job'), which removes the specified job and any subsidiaries from the job table, making the space which they occupy available to other jobs. The act of killing a job also releases any common heap space which it might have owned.

When this program has been assembled and saved to drive, possibly with the filename print_exec, it can be loaded and run at any time simply by typing EXEC mdv1_print_exec and pressing ENTER. To switch between active cursors on the screen, press CTRL-C. This will circulate around all the jobs awaiting keyboard input, but as you're only likely to have SuperBasic and this job resident when you first try it out, you only need one press of CTRL-C to switch between the two jobs.

Further issues of *E&CM* will contain many examples of multi-tasking QL programs, but remember the cardinal rule – ANY program which is self contained and does not require to do anything naughty, such as clearing the whole of RAM, will multi-task. It really is that simple.

Important things to remember are that when a job is first activated registers A4, A5 and A6 are set up to particularly useful values. A6 holds the address of the start of the job, (A6,A4.L) points to the start of the data area and (A6,A5.L) points to the end of the data area – that is, the end of the job. Also, remember that each job has its stack at the top of its data area – and this grows downwards!

When there are a large number of jobs running in a machine at any one time (and I've had 61 so far!), it would be useful to be able to have some degree of control over them from Basic. I've just written five procedures and one function to do just this, and they'll all appear in the next issue of *E&CM*.

"Any program which is self contained and does not require anything naughty, such as clearing the whole of RAM, will multitask".

©1984 Adam Denning

A change in the visible spectrum

Sinclair has given the Spectrum a facelift, but Gary Evans is disappointed to find that nothing inside the box has changed.

In a surprise, unSinclair-like move, an upgraded version of the Spectrum computer slipped onto the market in mid-October. The new machine, designated the Spectrum+, is hardware and software compatible with the 'old' computer, the major differences between the two models lying in the overall packaging of the product. At first sight the Spectrum+ looks like a stripped down QL and indeed it is evident that the new styling of the computer owes much to that of Sinclair's 68008-based superstar. Before getting down to a detailed assessment of the new 'go faster' Spectrum+ it would be as well to explain the statement at the start of this paragraph, namely that the new launch is a departure from the marketing approach that the company have followed in the past.

For the past three years Sinclair have launched computers that have been, each in their own right, quantum leaps in technology, the QL being the most recent of these. 1980 saw the launch of the ZX80 which was quickly followed by the ZX81 a year later. These two machines between them can justly claim to have set the home computing bandwagon rolling as far as this country is concerned. 1982 saw the arrival of the Spectrum, the computer that brought colour to the low cost computer market and again represented a quantum leap in both price and technical innovation. Finally, while just failing to make an '83 launch the QL continued the established trend. With the launch of each successive computer Sinclair, while not abandoning the earlier machines, have made it quite clear that as far as they were concerned, time had moved on and they were concentrating on researching new machines rather than refining existing models. With the launch of the Spectrum, the ZX81, a machine that had, and still has, plenty of life in it from a company point of view, was dead and buried. Now that pattern has changed and, although Sinclair now have a new glamour machine, they have decided to take another look at the Spectrum and to effectively relaunch the computer.

Rumours that Sinclair were considering the launch of an upgraded Spectrum have been rife since the introduction of the Timex engineered version of the machine in the States. This computer had a number of attractive features, real keyboard, on-board sound chip, a memory management system that allowed for bank switching and built-in joystick ports. Although the Timex computer failed to make any impact on the American market and has since been withdrawn, a few commentators thought that in the UK, Sinclair could make use of the R&D effort involved in the American computer in order to market an upgraded machine in this country. Those expecting Sinclair to take this route may be a little disappointed with the plus. Apart from taking the opportunity to tidy up the keyboard and to provide niceties such as cursor control keys, the only changes made are to the box in which the computer is housed.

In comes a QL style keyboard and indeed QL type case, the styling of which would suggest that a cut-down QL mould was used in its production. The case also incorporates a pair of feet that may be folded up in order to angle the keyboard to an ergonomically pleasing position. The attractions of this approach are that the Spectrum+ is completely compatible with all the vast range of hardware and software produced for the established computer.

"... a QL style keyboard".

The availability of an expansion pack comprising microdrive, Interface 1 and bundled software at under £100 is another boost to the computer's market appeal.

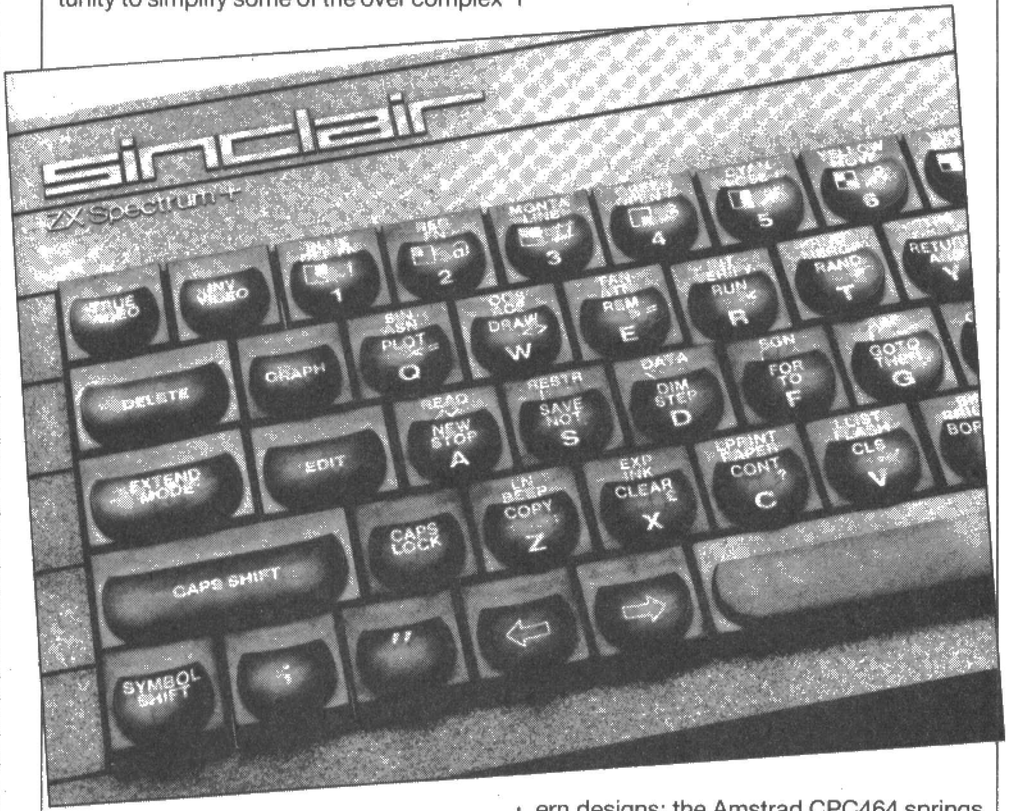
Having thus set the stage for the entry of the new computer, let's look at how it performs and decide whether the changes warrant the extra £50 or so that they add to the cost of the computer.

Something new

From the above it will be evident that much of the material with which reviews of computers normally concern themselves would be out of place in this assessment of the 'plus'. As far as the electronics and the firmware of the computer go, things are no different from the Spectrum that we all know and love. The powerful Sinclair Basic, the impressive graphics facilities and the not so impressive sound sections are all familiar. Even by today's standards the Spectrum is a powerful machine and the fact that the plus offers the facilities of the earlier version is not to say that it does not compete with some recent new models from other companies. The changes, as we have said above, concern the packaging. The QL-like keyboard is a vast improvement on the standard Spectrum, but will still not be to everyone's liking. The QL has come in for a fair degree of adverse comment when it comes to the quality of its keyboard. Many of these comments were unjust and can be put down to reviewers finding difficulty in saying anything intelligent about a computer that packed so much in the way of technical subtlety. The keyboard is not that bad, and while not up to the quality of some designs, even touch typists can, after a little practice, feel at home with it.

Apart from improving the action of the keyboard, Sinclair have taken the opportunity to simplify some of the over complex key assignments of the Spectrum. The symbols :"., along with all four cursor control keys are now available via a single key press. This also applies to a number of commonly used function keys. EDIT, DELETE, GRAPHICS and EXTENDed modes, CAPS LOCK, TRUE VIDEO and INV VIDEO all now have keys to themselves. The provision of a space bar is also a major step towards making the keyboard easier to use, although, in typical Sinclair fashion, the space bar is only half the length one would expect it to be.

Another welcome addition is the provision of a break key mounted to the right of the keyboard. This initiates a cold start and while its action achieves the same result as removing the computer's power, the break key is a more elegant way of terminating the action of a errant program. These changes make the plus a far more keyboard-friendly computer than its predecessor.



Mostly old

The changes to the design of the keyboard and case are as far as Sinclair have gone. The only I/O that the plus provides is the familiar Spectrum user port, in effect an unbuffered port making available the internal data, address and control lines of the computer. It would have been nice to see a printer port, joystick port(s), perhaps a partially decoded user port, even an explicit RGB or composite video output (such signals are available at the user port but a separate connector would have made them more accessible to the majority of users). Provision of some, even all, of these facilities would not have been a major cost and would have added greatly to the machine's appeal and brought it into closer competition with some more modern designs; the Amstrad CPC464 springs to mind. As it stands it is a little difficult to see where the extra £50 or so that the public are being asked to pay for the plus, as opposed to the basic machine, has been spent. All the more difficult when the fact that much of the production engineering costs involved in the new case must have been shared with the development of the QL's casing.

Screen shot

The manual that accompanies the Spectrum+ is a dramatic departure from the user guide of the basic machine. Gone is the spiral bound volume of black & white pages, in its place comes a colourful tome including many screen shots illustrating the output of the numerous example programs. The manual was produced for Sinclair by Dorling Kindersley and adopts the style of the company's excellent 'Screen Shot' series of publications. The tutorial tape that accompanys the plus was produced by Goldstar, an off-shoot of DK and this too is an excellent introduction to the new computer.

The familiar collection of software represented by the Spectrum six-pack is also part of the plus package. This represents a good selection of software, most of it well written and good value although, with the best will in the world, the 'make a chip' package can only be described as a space filler in the six-pack packaging.

Conclusion

Will the new version of the Spectrum make any impression on the market? Well, the answer is likely to be yes but not that much. The souped-up version of the computer is without doubt an improvement on the basic machine but whether or not the improved performance is worth paying £50 for will depend on the applications to which the computer is to be put. There is little point in games players paying extra money for the plus, most of them will want to use joysticks with the machine and will not see the improved keyboard as necessary. Only those users who foresee a need for entering considerable amounts of text/data to the computer are likely to consider the plus. Such applications would include programming or text processing. Existing owners of Spectrums with this pattern of computer use will, in all probability, have purchased add-on keyboards,at any rate the plus in not aimed at existing owners wishing to upgrade. The Spectrum+ must appeal to first time computer buyers of a more serious nature. To this group of people the lack of any significant I/O provision could be a disadvantage.

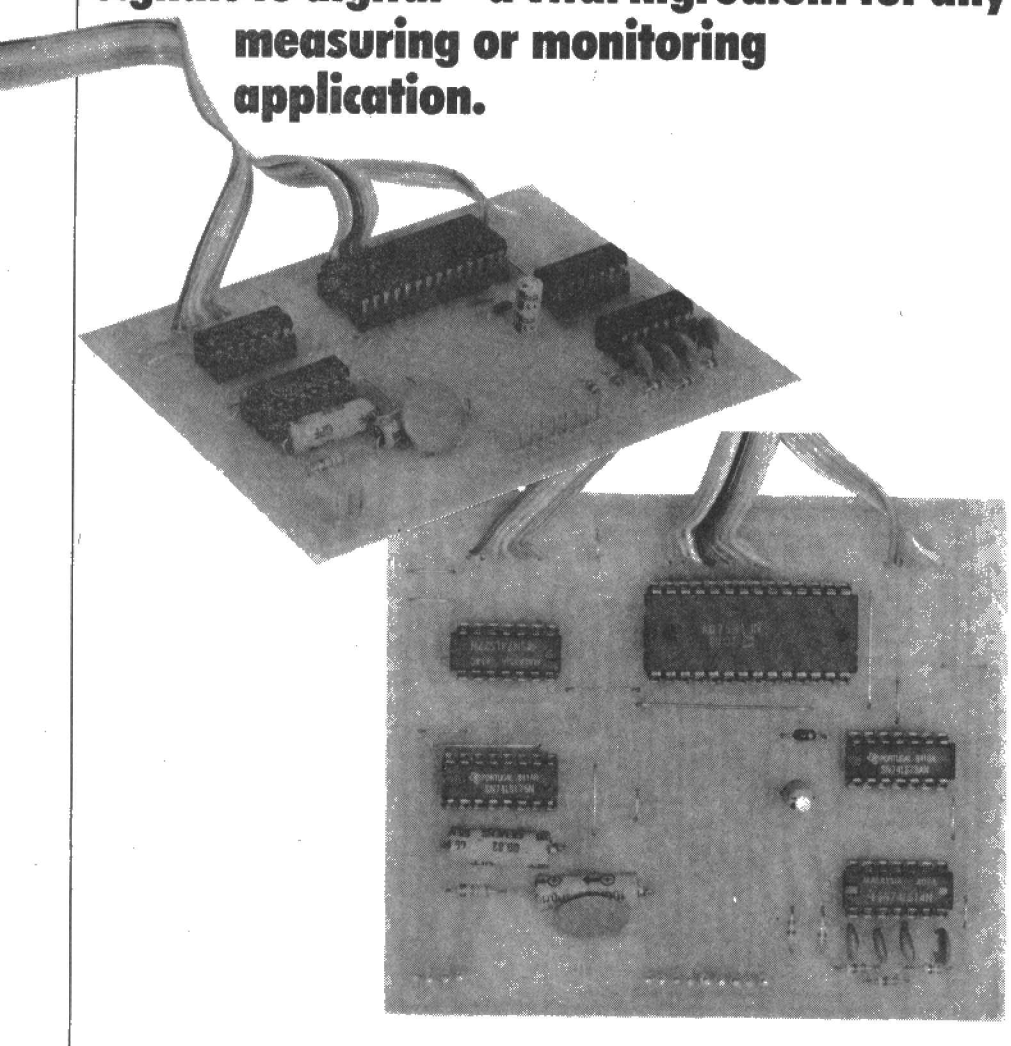
The major plus for the plus is that there is already a vast software base for the Spectrum and the new machine is fully compatible with this. A very powerful argument for anyone considering the purchase of a computer.

On the whole the plus is a rather disappointing computer in terms of the very high standards Sinclair have themselves set over the years. There seem to have been many missed chances in rethinking the design of the computer. In the words of a Sinclair spokesman, "the Sinclair+ represents a marketing innovation". This is does, but it is a pity that this time around there is no technical innovation to accompany it.

"The Sinclair plus is a marketing innovation – it is a pity that this time around there is no technical innovation to accompany it".

Amstrad A/D convertor

Robert Penfold hangs a board on the Amstrad CPC464 to convert analogue signals to digital – a vital ingredient for any measuring or monitoring application.



One of the most useful general purpose add-ons for any computer is an analogue to digital converter. This enables a computer to be used easily in practically any measurement or monitoring application, including such things as temperature, and moisture measurement. An interface of this type becomes even more useful if it also has one or more digital outputs, as applications which involve control in addition to measurement then become possible. For example, instead of simply acting as, for example, a thermometer to monitor something like a photographic solution or a greenhouse, with suitable hardware the system could be used as a temperature regulator. The software can be very simple, and is usually just of the IF...THEN variety, with something like an output being set high if the returned value is below a certain level, or set low if it is above a certain figure.

This 8 bit analogue to digital converter for the Amstrad CPC464 computer has 8 inputs, with an input voltage range of 0 to 10 volts at each input. Additionally it has a nibble (4 bit) output that provides standard LS TTL latching outputs. With the aid of this unit the CPC464 should therefore be readily usable in the vast majority of measurement and control applications with a minimum of additional hardware. The converter is not a high speed type, with each channel being converted at 640us intervals, or about 1562 times per second in other words. This is not adequate enough for a few specialised applications such as audio digitising, but it is more than sufficient for most applications. It is certainly fast enough when using Basic, since conversions are produced more rapidly than the time it takes even just a single channel to be read.

The unit connects to the "floppy disc" port of the computer, which is actually a general purpose expansion port that is ideal for user add-ons.

System operation

The block diagram which appears in **Figure 1** shows the overall make-up of the unit. The 8 x 8 RAM, analogue to digital converter, 8 channel multiplexer, and control circuit are all part of the 7581 integrated circuit which is used as the basis of the unit.

Although the 7581 is a complex chip, the way in which it is utilised is reasonably straightforward. Each channel is automatically read in turn by the successive approximation analogue to digital converter, and the returned values are stored in the 8 x 8 RAM (one 8 bit RAM for each channel). By reading the appropriate section of the RAM the latest reading for the required channel is obtained. This system is perhaps a little less versatile than some of the alternatives, but it is delightfully simple in that it avoids the need for any form of handshaking between the converter and the computer.

An external clock signal of around 1 to 1.5MHz is needed to control the timing of the chip, and in this case a 1MHz clock signal is obtained by dividing the 4MHz system clock by four. The converter section of the 7581 needs a –10 volt reference source, but the expansion port of the CPC464 does not have a negative supply output. The only supply that is provided is +5 volt type, and the –10 volt reference therefore has to be generated from this in some way. The method used here is to take the 1MHz clock signal, and to rectify and smooth it using a voltage multiplexer circuit. A simple regulator circuit is used to stabilise the resultant negative supply at the required potential.

An address decoder drives the output enable input of the 7581 and places the circuit in a vacant area of the input/output map of the computer (&F800 to &F807). The nibble output is provided by a four bit data latch, and this output port appears at the same addresses as the converter.

The circuit

The full circuit diagram for the unit is given in **Figure 2.** The CPC464 has a Z80A microprocessor using a 4MHz clock, and the 7581 will interface to this without difficulty. The method of input/output mapping in the CPC464 is a non-standard arrangement which consists basically of having one of the eight most significant address lines going low to activate an input/output

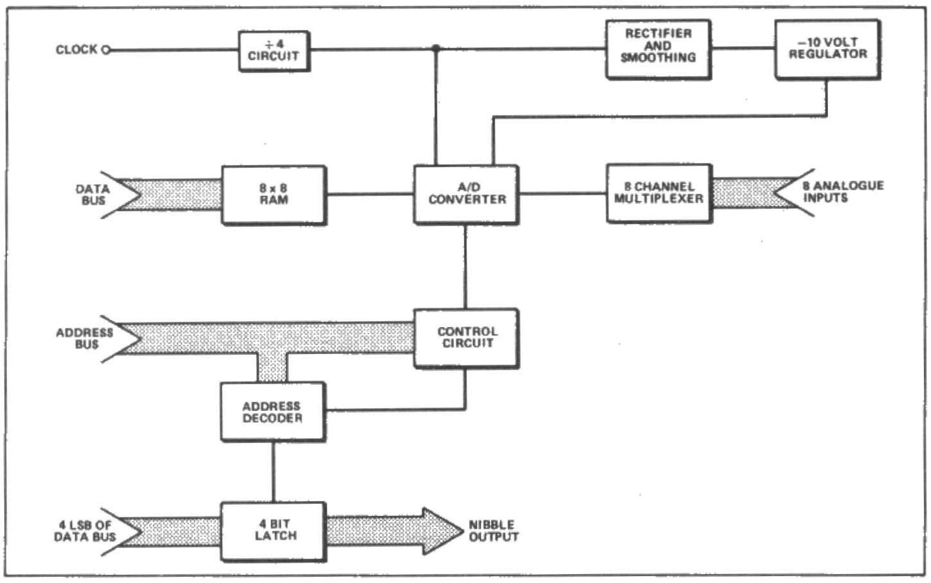


Figure 1. Block diagram of the CPC464 8-channel ADC.

device. Where a device requires more than one address, then more of the address lines, including the eight least significant lines, can be used as well. For external add-ons A10 is available.

The data bus of the 8 x 8 RAM has three-state outputs which connect direct to the data bus of the computer. The 7581 has three address inputs which are used to select the desired channel, and these are connected to the three least significant bits of the computer's address bus. This conveniently places the converter at eight consecutive addresses. The enable input of IC1 is driven from an address decoder circuit which is formed from two 3 input NOR gates (IC2a and IC2b). This gives a negative enable pulse to IC1 when the RD, A10, and IORQ lines are all low. As mentioned earlier, this places the converter at the eight addresses from &F800 to &F807. The very simple system of input/output mapping utilised in the CPC464 means that there are numerous addresses for each channel of the unit, but in practice it is best to use those mentioned above as they are quite convenient and avoid spurious operations of other input/output circuits.

Pin 1 of IC1 connects to the input of the converter via an internal resistor. This terminal is connected to ground so that the quiescent input voltage is zero. This resistor sets the input resistance of each channel at about 20K. The input resistance is extremely high if pin 1 is left unconnected. The 7581 has separate analogue and digital ground terminals (pins 11 and 14 respectively), but in this circuit they are both connected to the 0 volt supply rail of the computer. The 7581 has a status output at pin 12 which pulses low each time a conversion is completed, but this is only needed in applications that use interrupts, and this output is not implemented here.

IC3 is a dual JK flip/flop, but here both sections are connected to operate as simple divide by two circuits. They are connected in series to give a divide by four action, producing a 1MHz output from the 4MHz system clock input signal. The Q output is used to drive the clock input of IC1 while the negative Q output drives the voltage multiplier circuit. The latter is a conventional four stage type which uses

"An interface of this type becomes even more useful with one or more digital outputs".

four Schmitt Trigger/Inverters from IC5 as buffer stages. In theory the circuit is a voltage quadrupler, but in practice, inefficiencies in the circuit reduce the final output voltage to only about −12 volts or so, but this clipped at the required −10 volts by zener diode D6.

IC4 is a quad D type flip/flop, and in this circuit it operates as a 4 bit data latch with the latching pulse from the address decoder fed to the clock pulse input. R1 and C3 provide a negative reset pulse at switch-on. The four data inputs are fed from the four least significant bits of the data bus. The address decoder for the nibble output is comprised of IC2c plus IC5d, and these generate the negative latching pulse when A10, IORQ, and WR all go low. Separate address decoders for the converter and nibble output (one gating the RD line and the other gating the WR line) enables the two ports to share the same address range without operation of one causing spurious operations of the other. Like the converter, the nibble output occupies a vast address range, but in practice it is advisable to operate it using an address in the range &F800 to &F8FF to avoid unwanted operations of the computer's internal hardware.

Construction

Refer to **Figure 3** for full details of the printed circuit board. As the 7581 is a fairly expensive device and is a CMOS type it would not be sensible to ignore the standard MOS antistatic handling precautions. Use an integrated circuit holder for this component, leave it in its antistatic packing until it is time for it to be plugged into position, handle it as little as possible, and do not fit it onto the board until construction is in all other respects finished. Be very careful to fit IC1 and the other semiconductor components onto the board the right way round.

The component panel connects to the floppy disc port of the computer by way of

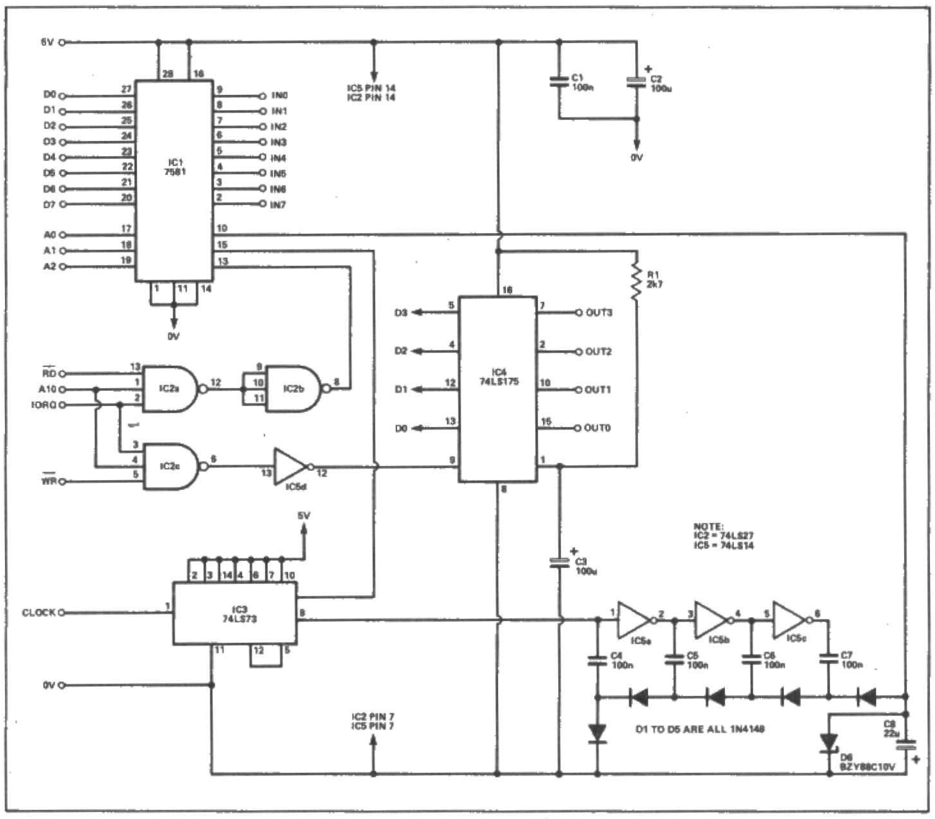


Figure 2. Full circuit diagram including the nibble output.

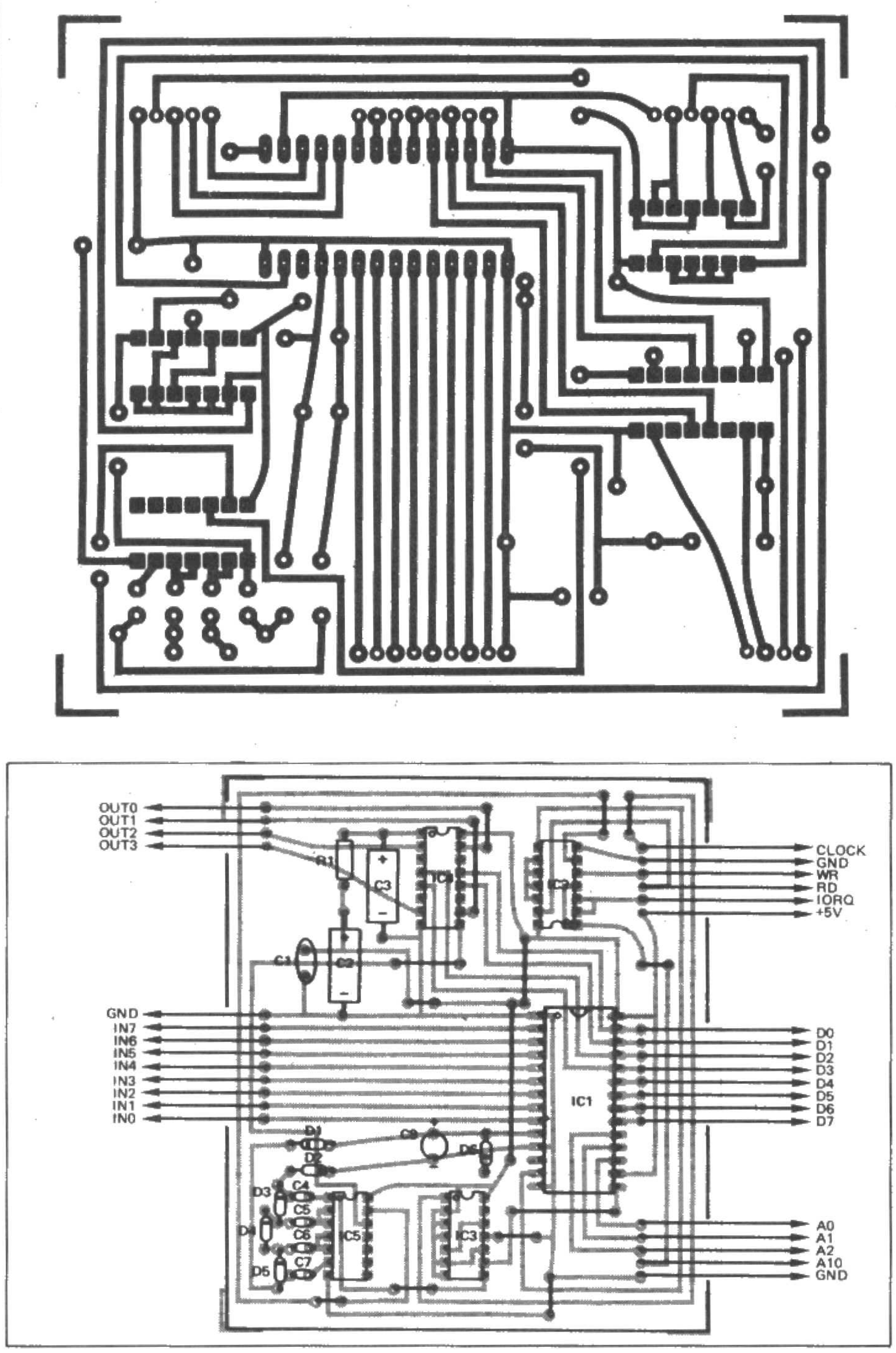


Figure 3. The printer circuit board foil and overlay.

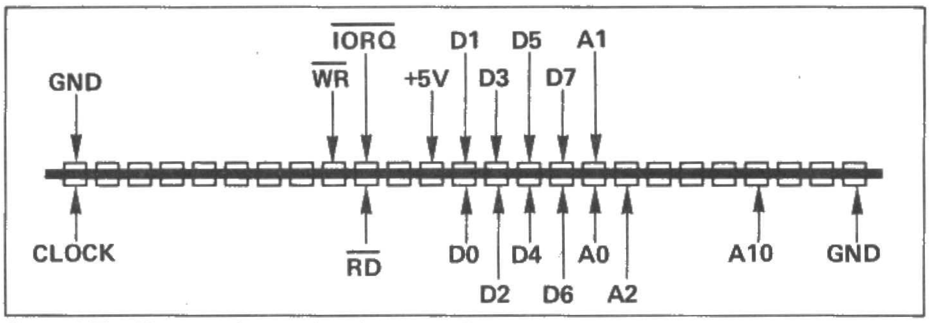


Figure 4. The 19 connections to the edge connector.

a piece of 19 way ribbon cable of up to about 1 metre in length. The free end of this cable is fitted with a 2 by 25 way 0.1 inch pitch edge connector. Connection details for this are shown in **Figure 4.** The floppy disc port of the computer has provision for a polarising key, but it is unlikely that a connector fitted with a suitable key will be available. With a little ingenuity it might be possible to add a suitable key, or failing that the top and bottom edges of the connector should be clearly marked.

Thoroughly check the finished unit for errors, and in particular check the connections from the board to the edge connector. It would not be difficult to get a crossed wire here, and "rainbow" ribbon cable is probably the best type to use as this helps to minimise the risk of a misidentified lead.

Testing

Connect the unit to the floppy disc port of the computer before switching on. After switch-on the usual screen display should be produced – switch off immediately and recheck the unit if it is not. Assuming that the computer operates normally, the command:

```
OUT &F800,15
```

should set all four lines of the nibble output to the high state. The command:

```
OUT &F800,0
```

should then return them to the low state. As only the four least significant bits are implemented, the number written to the port is, of course, in the range 0 to 15 and not the usual 0 to 255. Note also that as the Z80A microprocessor has separate memory and input/output maps, data transfers are accomplished using OUT and INP rather than PEEK and POKE.

Channels 0 to 7 of the analogue to digital converter are at addresses &F800 to &F807 respectively. Reading any channel with no input connected should return a value of zero. As a simple check of the unit connect input 0 to the +5 volt rail (the positive leadout write of C2 makes a convenient connection point) and then take a reading for this channel using the command:

```
PRINT INP(&F800)
```

With a full scale value of 10 volts and the supply potential at 5 volts, if the unit is working properly the returned value should obviously be about half the full scale value (ie something in the region of 126 to 129). A more comprehensive check could be made using, for example, a 9 volt battery and a potentiometer of about 10k in value to give a range of test voltages.

PARTS LIST

Resistor (¼ watt 5%)

R1	2k7

Capacitors

C1,4,5,6,7	100nF miniature ceramic (5 off)
C2,3	100uF 10V axial elect (2 off)
C8	22uF 16V radial elect

Semiconductors

IC1	7581
IC2	74LS27
IC3	74LS73
IC4	74LS175
IC5	74LS14
D1 to D5	IN4148 (5 off)
D6	BZY88C10V

Miscellaneous

Printed circuit board
28 pin DIL IC holder
16 pin DIL IC holder
Three 14 pin DIL IC holders
2 by 25 way 0.1 inch pitch edge connector
Ribbon cable, wire, solder, etc.

INTERACTIVE VIDEO

The launch of Acorn's interactive video system based on the BBC micro has prompted Richard Sargent and Peter Luke to investigate the technology behind the concept.

The launch of Acorn's Interactive Video System was an occasion more memorable for the degree of hype evident than for an innovative product making its debut. While there is little doubt that the fusion of the technologies of the computer and video disc industries will have a great deal of impact on the future of information technology, it will be quite some time before some of the applications conjured up by Acorn Video have any great impact on the lives of the public at large.

So just what is interactive video and why are there such high expectations as to the future impact of the concept? In order to appreciate what the new technology is all about we must first explain something of the technology of the video disc.

Disc O Hi-tech

A video disc is the name given to any disc, made of whatever medium, which is capable of storing pictures. These pictures may be moving or still, the recording technique may be analogue or digital.

Now, this crisp definition of what a video disc may (or may not) be has become somewhat blurred in recent years as manufacturing companies vie with each other and make vague (but impressive) claims as to the technological brilliance of their system X. So it might be as well to start at the beginning and trace the heady rise of this medium.

A short history

Video disc experiments have spanned three continents; America, Europe, and Asia, represented by Japan. The early pioneers were Decca and Telefunken and their efforts were based on the then-prevalent technology of vinyl discs and spiral grooves. The system did at least work, although recording lengths were very short, and extensive wearing of both stylus and disc prevented any long-term hope of success. At about the same time (the early sixties) the American RCA and Zenith corporations were working together on another spiral-groove system which was less subject to wear. The stylus and disc surface together formed a capacitor, the charge on which varied as the stylus – which was one of the capacitor plates – passed over indentations on the disc surface – which represented the other capacitor plate. Thus the Capacitance Electronic Disc (CED) system was born, and it's still around today, albeit in a much refined form.

The 1970's saw the beginning of experiments in optical recording and playback techniques using low-powered lasers. Several systems have been tried, including passing light through a transparent "floppy" disc, but it was reflected-light systems which proved to be the better bet and the first of these was demonstrated by Philips at Eindhoven in 1972.

Optical discs

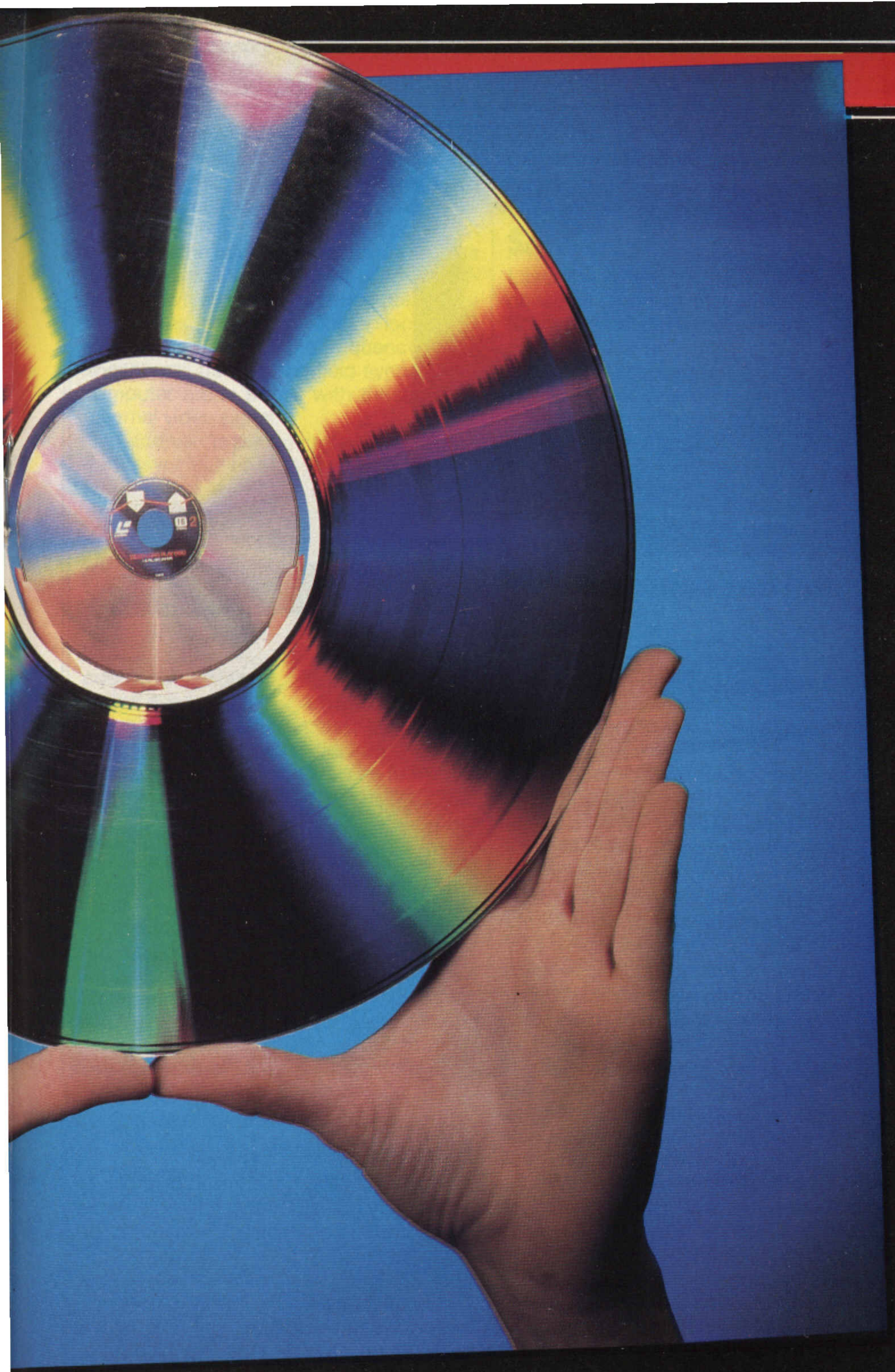
The optical disc, or LaserVision as it was quickly christened by Philips, has had a long and stormy development. The official launch to the American consumer in 1970 brought a less than overwhelming response, and you might be forgiven for not noticing the European launch in 1982. The Japanese, not to be outdone, have Pioneer working on the LaserVision system, while JVC have perfected a CED-type disc known as VHD.

"Computers are looking toward the video disc with a view to partnership".

Today in Britain you can buy a Philips or Pioneer LaserVision player, or a Thorn-EMI VHD player. You won't have a very wide choice in the type of film you can buy and play on your new machine, but all that is going to change over the next few years. Computers are looking towards video discs with a view to partnership, and when this really takes off it's going to make present day computer peripherals look insignificant.

Not the sound of music

Video discs usually play for 60 minutes a side, so all but the longest films can be squeezed onto the two sides of the one disc. The discs have a spiral track (but *not* a spiral groove) some twelve miles in length running from perimeter to centre, a diameter of 10" in the case of VHD discs and 12" in the case of LaserVision discs.



Both systems have all the analogue information on colour, luminance, sound (two channels) and video control recorded by frequency-modulation on the track surface.

LaserVision discs

A track on an optical disc is made up of a series of micron-sized pits which are illuminated by a low-power helium-neon laser and read back by a photodiode. Each pit varies in length, this being proportional to the analogue signal encoded in it; but the width is constant at 0.6um and so is the depth, at 0.1um. A gap of 1.6um separates one track from another, and there are 54,000 tracks in all. The optical pulses from the micro pits are demodulated to produce both video and audio carrier signals, which are subsequently converted into pictures and sound. It goes without saying that the LaserDisc player must be build to extraordinarily fine engineering tolerances – there are, for example, no grooves for the photodiode assembly to follow, so servo motors guided by sophisticated feedback electronics must track the arm across the disc surface. The laser beam must have a short wavelength to focus on such narrow tracks, and cheap solid state lasers are not yet sufficiently reliable to undertake this task. Current players use the more expensive helium-neon gas lasers. However, the absence of grooves and stylus together with the presence of a finely controlled tracking arm does mean that the readhead can zip from one part of the disc surface to another. In terms of computer jargon, the video disc has a Random Access capability. This has far reaching implications which have very little to do with films-on-disc, but which have a great deal to do with computer controlled pictures-on-disc. A LaserVision disc can store 54,000 separate picture frames on each disc surface, and it does so when allowed (by design) to rotate with constant Angular Velocity (CAV).

The CAV format disc (see **Figure 1**) has its recording arranged so that one complete television frame (two fields) is placed on each track. Hence each revolution represents one video frame and the speed of rotation is synchronised with the standard television framing rate, giving a disc speed of 1500rpm. In this mode the information is packed more tightly at the centre of the disc than on the outside. This arrangement makes it possible to achieve perfect still-frame and slow-motion replay by reading the same track more than once, the read-head jumping back across the spiral during the vertical synchronisation interval when no picture information goes to the television screen. The drawback with this format is that it only gives 37 minutes of playing time per side. So, if you want to see a film from start to finish on the one disc, it needs to have been recorded at Constant Linear Velocity (CLV). Here the information is packed with the same density all over the disc, and the speed of rotation actually varies to compensate for this, slowing down whenever information at the periphery of the disc is read. In CLV mode, still frame and slow motion are far from perfect, but playing time is back to one hour per side.

"Laser discs are the natural choice for permanent storage".

The VHD disc is in many ways quite similar to the LaserVision disc in terms of the end result; it has a CAV mode, numbered frames (45,000) and still-frame facility. Yet there are perhaps two physical characteristics that make it the poor relation to the optical disc. Firstly it has a stylus – albeit a very light one – so that disc wear and degradation of picture cannot be discounted especially after the frequent use of the still-frame facility. Secondly, the disc itself is rather fragile. This in itself is not of great importance, after all floppy discs are very fragile and nobody seems to mind very much, but when you consider the LaserVision disc has a surface insensitive to scratches, dust and grease, it is perhaps all too obvious which would be the natural choice for permanent, archival material.

Coding the discs

If you have a random access disc the ability to pick and choose your tracks is only as good as the addressing system used – you need to be able to tell the tracking arm where to go. On LaserVision discs coded information is recorded prior to each picture frame. CAV format discs have the most room at their disposal for this extra information, and those are, of course, the very discs which benefit from the additional information. Each of the 54,000 pic-

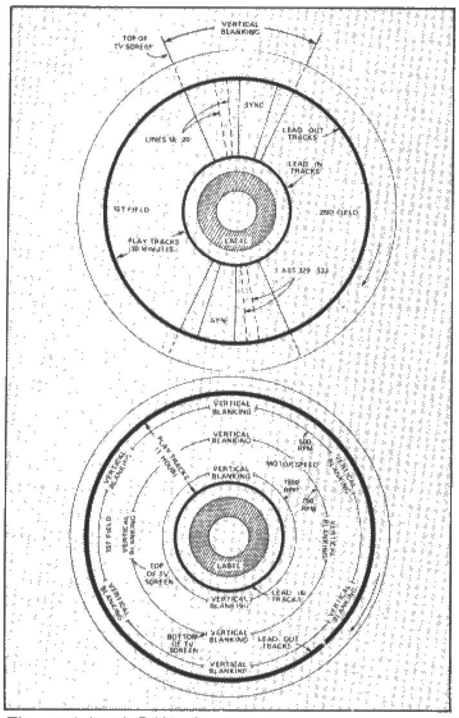


Figure 1 (top) CAV disc format.
Figure 1 (below) CLV.

ture frames may be given an absolute code number, and "page" and "chapter" numbers may also be allocated.

Teletext-style information can be laid down on the disc, so that screenfuls of text do not constitute a picture frame but are available in addition to the pictures. What we have here is a system known as Active Video.

A computer can control the video player (since it can send the tracking arm to any one of 54,000 frames on the disc), but the computer itself needs to be controlled by a program. This program can be in ROM, or it can be on floppy disc. It could also be on the video disc, in the form of raw data to be down-loaded into computer memory and, once there, acted upon. But most fascinating of all, the Computer-Video Disc set up can be made to be interactive. Here, the viewer becomes an active participant in the programme he or she is watching.

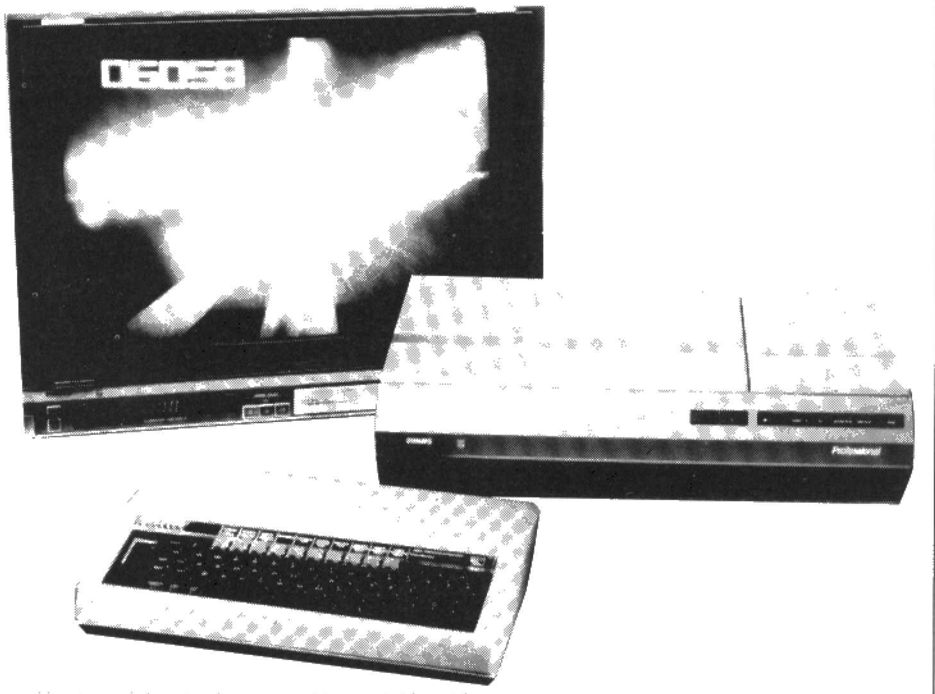
All that gen

Another important element in an interactive video system is the video interface between the text/graphic signals produced by the controlling computer and the high resolution images being produced by the video disc player. In some systems separate monitors are used for the two displays but this goes against the concept of the integrated system and Acorn, in common with other people in the field, have made provision for the two video signals to be combined in a single display. To achieve this aim is not quite as straightforward as it may appear at first sight.

The problem lies in synchronising the signals produced by the computer and disc player. While not wishing to delve too deeply into the subject, any video mixing system must provide a means by which both the line and frame sync pulses of the two video signals to be mixed can be brought into phase. A number of techniques to achieve this goal have been developed by the video industry over the years. These include locking all video sources to a central sync pulse generator (this approach is used in commercial TV studios) to digital frame stores that capture video signals in one half of their circuit and are then able to digitally play these out in step with an external video signal in another section of the unit. These techniques, particularly the digital frame store are all, however, very expensive solutions to the problem of syncing video signals.

Fortunately there is another technique that can be applied in the case of the video signal produced by a computer. This relies on the fact that, in the majority of computer designs all the timing of the various signals

"The computer/video disc set-up can be made interactive. Here, the viewer becomes an active participant"



Acorn Interactive Video

Acorn Video, the new wholly-owned subsidiary of Acorn Computers, recently launched their first system. The product of a two year development programme, the system comprises of a modified BBC micro, dual disc drives, modified monitor, LaserVision disc player and EPROM based controlling software. The micro is connected to the disc player by way of an RS232 link. This allows the computer to control all of the functions of the video disc player. At present the link is uni-directional but future systems could make provision for bi-directional communication thus allowing software to be uploaded from the video disc as described in the main body of this article.

At present Acorn Video would appear to be concentrating on the vast industrial training market. Such users will be able to obtain complete systems for a cost in the region of £3000 to £4000. At present there are no specific plans to operate an upgrading service for those organisations that will already have many of the components of the system to hand. According to Acorn Video, this is because of the large number of BBC micros in the field, only the very latest versions are easily modified for AIS operation. No doubt though that if there is sufficient demand from educational establishments, Acorn may well reconsider this part of their plans. Even allowing that the micro itself can be converted there is still the problem of the requirement for a modified monitor in order to use the system to its full potential.

Philips LaserVision

The design of the LaserVision disc player was largely due to Philips, a familiar name in the consumer electronics market. The company market a range of players for both the home and professional markets. The system has not to date lived up to initial expectations in terms of the market penetration expected to be achieved. Interactive Video should provide a well deserved boost to the sales of players though.

inside the machine is derived from a master clock. Altering the frequency of this can alter the phase relationship between any computer generated signal and an external source. To convert a computer for gen lock operation the usual approach is to replace the master crystal with a collection of components which include a substitute crystal that forms part of a phase lock loop system. This will pull the frequency of the crystal until the line and field syncs of the computer are in phase with those of the external video signal.

In addition to ensuring that the line and frame sync pulses of the two video signals are in phase with each other there is the additional problem of synchronising the colour subcarriers of the two signals. This is achieved by an extension of the application of the phase locked system described above. The variations in the frequency of the master clock are so slight that they have no noticeable effect on the operation of the computer.

The signal produced by the gen lock board will most often be in the form of a composite video signal. This is not the case in the Acorn design, however, as this system produces an RGB output plus one other line to tell a specially-converted monitor when to switch between the two video signals.

Software scarcity

LaserVision was originally conceived as a consumer product that would provide an alternative to the video recorder. Despite the fact that research shows that the majority of video recorders are mainly used to play back pre-recorded material, the public consider the ability to record more important than the improved image quality of the laser-based players. Therefore to date sales of video disc players have been disappointing. The low sales have in turn meant that there has been little incentive for publishers to make material available in this medium.

"The computer program can prompt the user to make a response, as a result of which the video can jump to alternate sequences".

The interactive video industry will thus need to concern itself with the production of disc-based software. Many people have the idea that it costs a small fortune to produce video discs in small quantities – fortunately this is not the case. Of the total cost of any disc, by far and away the greatest cost is in the production of the images that are to go onto the disc, the master 1" video tape.

The cost of mastering the disc is, in comparison, quite small. To master one side of a disc of between 16 and 36 minutes costs in the region of £2000. Pressing costs fall to around £10 for runs in the range of 200 units. These figures mean that discs can be produced at a resonable cost even where the numbers of copies required are not in the 1000's.

Another drawback of the laser disc is that it is a read only medium. The players would be far more versatile if some form of write (even quite a limited capability) could be built in. This development, while not just around the corner, will be here within a couple rather than tens of years. This will greatly increase the potential of interactive systems.

The computer program, be it educational, instructive, or purely recreational can prompt the user to make a response, and as a result of that response the video can jump to alternate sequences, or repeat past sequences, freeze a frame, or replay backwards in slow motion. The human participant may be learning in Arabic how to erect a desalination plant, or may be revising for a Biology "O" Level, or even pretending to fly a spacecraft through the Rings of Saturn – in any event as film technology converges with computer technology, you can be sure that the video disc is likely to be around in some shape or form, and in very great numbers, for a long time to come.

E&CM PCB SERVICE

October 1983	
Cassette Signal Conditioner	£1.60
BBC EPROM Programmer	£6.66
November 1983	
Lie Detector Interface	£2.45
Microcontroller	£2.77
ZX Light Controller	£5.56
December 1983	
BBC Sideways RAM	£6.48
Electron A/D	£3.78
January 1984	
Electron I/O Port	£3.02
February 1984	
BBC Speech Synthesiser	£5.89
Electron RS432	£3.51
Spectrum Speech Board	£4.18
BBC Sideways ROM Board	£7.13
March 1984	
Spectrum Cassette Controller	£2.59
April 1984	
Commodore A/D	£2.15
May 1984	
Memex	£7.55
Spectrum Diary	£4.26
Centronics Buffer	£7.41
June 1984	
Mains Data Link (2 Boards)	£4.72
July 1984	
IR Data Link (2 Boards)	£3.95
August 1984	
Robot Wall Builder	£2.70
September 1984	
Spectrum Frequency Meter	£3.61
October 1984	
EPROM Simulator	£5.85
November 1984	
Amstrad PIO	£5.65
December 1984	
Amstrad CPC464 A/D	£4.10

HOW TO ORDER

List the boards required and add 45p post and packing charge to the total cost of the boards. Send your order with a cheque or postal order to:

E&CM PCB Service, Scriptor Court,
155 Farringdon Road, London EC1R 3AD
Telephone: 01-837 6255

Please supply the following PCBs:

....

....

Post & Packing 45p

TOTAL £

Signed .... Date ....

Name .... (please print)

Address ....

....

....

PLEASE ALLOW 28 DAYS FOR DELIVERY

Floppy Turnover

A mere two minutes snipping and punching will transform a humble single-sided floppy into a double-sided disc. Chuck Shenton explains this must for all misers.

In spite of their popularity, single-sided drives have one major limitation; they only allow half of the potential disc capacity to be used. This is because the drive's single read/write head can only access the lower side of the disc, leaving the upper side, which is also capable of storing data, totally wasted.

As it name suggests, the double-sided drive overcomes this limitation by incorporating a second head for the upper surface but, although this has the added advantage that both sides can be used at the same time, there is a penalty to be paid in terms of the cost and complexity of the drive.

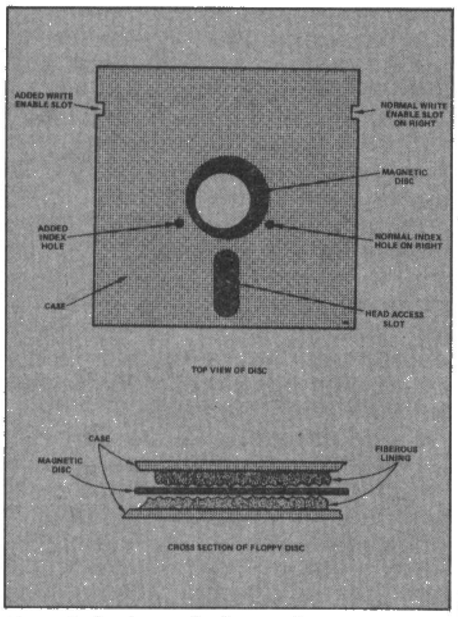


Figure 1. Anatomy of a floppy disc.

It is not feasible to simply invert a single sided drive because, although the head slot would align with the head, the write enable slot and index hole in the disc case would not line up with the drive's sensors.

However, by following the procedure described in this article a second write enable slot and index hole can be cut into the case to make the disc symmetrical; it will then operate correctly when inverted.

To modify a disc the following items are required: a scalpel or craft knife, large needle, protective envelope, cardboard template, and modified hole punch.

The scalpel is used to cut an extra write-enable slot in the case.

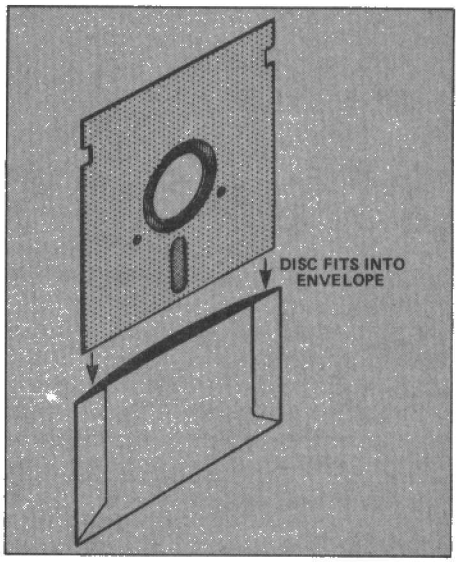
The protective envelope can be made from stiff paper as shown in **Figure 2,** and is used to cover the head access slots and reduce the chance of damage to the active surface while the disc is being modified.

The template simplifies the marking out operation and is made to the exact pattern shown in **Figure 3**; any thin stiff cardboard will do.

The best approach is to transfer the pattern on to the cardboard with carbon paper and a sharp pencil. Then, using the needle, pierce the template master and cardboard in the positions for the left and right index holes.

The outline is then accurately cut out with a pair of scissors, leaving the write-enable slots and centre drive hole to be removed on a flat surface using the scalpel. This completes the template (but you can also make life easier by marking the top with a felt tipped pen).

The punch, is used to form the extra index holes in the case and is central to the



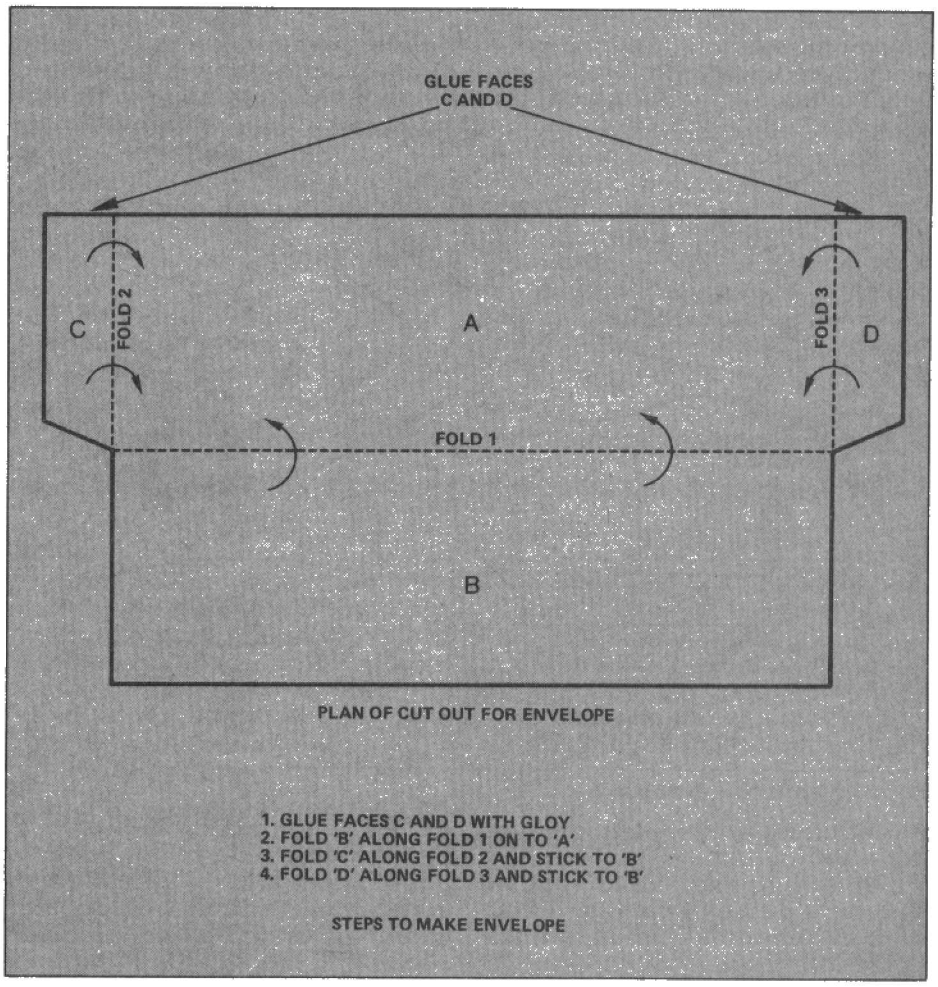
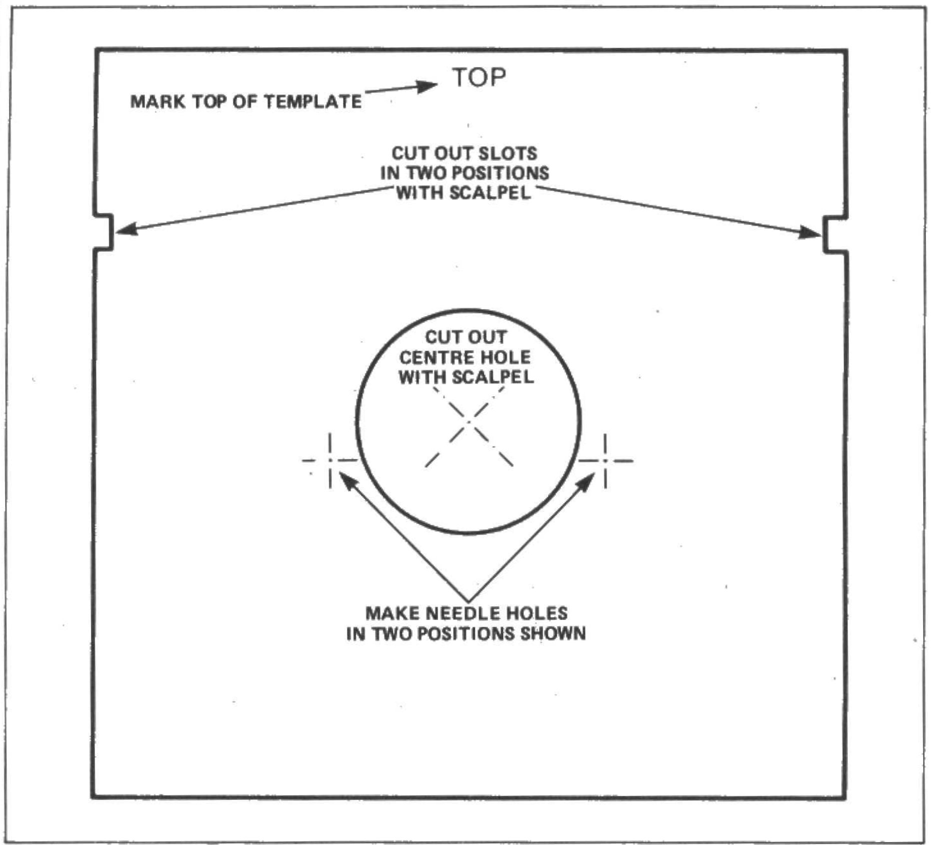

GLUE FACES C AND D
C
FOLD 2
A
FOLD 3
D
FOLD 1
B
PLAN OF CUT OUT FOR ENVELOPE
1. GLUE FACES C AND D WITH GLOY
2. FOLD 'B' ALONG FOLD 1 ON TO 'A'
3. FOLD 'C' ALONG FOLD 2 AND STICK TO 'B'
4. FOLD 'D' ALONG FOLD 3 AND STICK TO 'B'
STEPS TO MAKE ENVELOPE

Figure 2. The protective envelope.



MARK TOP OF TEMPLATE
TOP
CUT OUT SLOTS IN TWO POSITIONS WITH SCALPEL
CUT OUT CENTRE HOLE WITH SCALPEL
MAKE NEEDLE HOLES IN TWO POSITIONS SHOWN

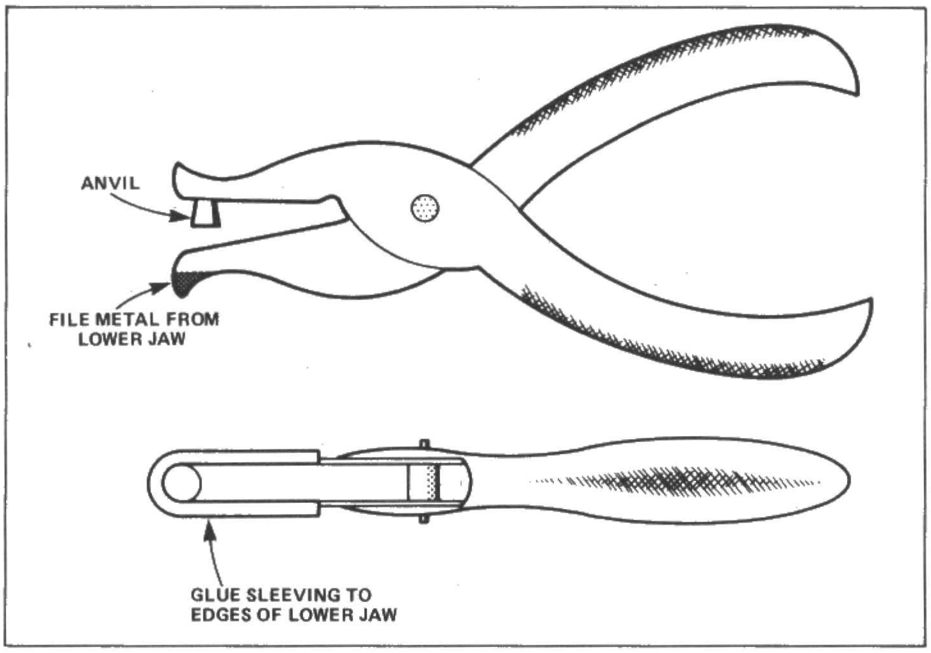
Figure 3. Plan of template (75% size).

whole operation.

My first attempt at making these holes entailed removing the magnetic disc and cutting square holes in the case with a scalpel. Surprisingly the disc worked perfectly after this, but the result looked rather untidy and the entire business took too long anyway.

It became obvious that to make the operation a practical proposition a ready made tool would be essential, but after looking at various devices it seemed that nothing suitable was available. However the answer finally came in the form of a neat, single hole stationery punch, sold by W. H. Smith for the modest sum of £1.30.

The punch could be used at it stands but the lump on the lower jaw would make this awkward and may also lead to distortion of the case and damage to the disc. To avoid this the punch should be modified as shown in **Figure 4.**



ANVIL
FILE METAL FROM LOWER JAW
GLUE SLEEVING TO EDGES OF LOWER JAW

Figure 4. Modifications to the punch.

First, as indicated, file the lump off the lower jaw, taking care to remove any rough edges. Next, using the scalpel, slice a short piece of PVC sleeving along its length and glue it on to the bottom edge of the jaw with impact adhesive.

Marking out

To avoid contamination of the disc it is essential that all operations are carried out in a clean and dry area. Fit the protective envelope then mark out the case as follows:

Accurately align the top of the template with the top of the disc then, using a sharp HB pencil or fine felt-tipped pen, mark out the position for the extra write-enable slot on the right.

Push the needle through the index needle hole on the right of the template and lightly mark the case. Take great care to apply the minimum amount of pressure necessary to mark the case and on no account puncture it.

Turn the disc over, left to right, and carry out the same operation but this time use the left index needle hole in the template.

Modifying the disc

Rest the disc on a flat surface and, using the scalpel, cut out the extra write-enable slot.

Next, at the periphery of the drive hole, gently lift the case away from the magnetic disc and, with great care, insert the bottom jaw of the punch. Then, after ensuring that the anvil of the punch is concentric with the needle mark, punch out an index hole in the top of the case. The hole need not be in exactly the right position, but be as accurate as you can.

Punching out the index hole is the most tricky operation, but with a little practice, most people should find it well within their capabilities. However care must be taken not to damage the case or the magnetic disc and never insert the jaw of the punch in far enough to come into contact with the active part of the disc surface.

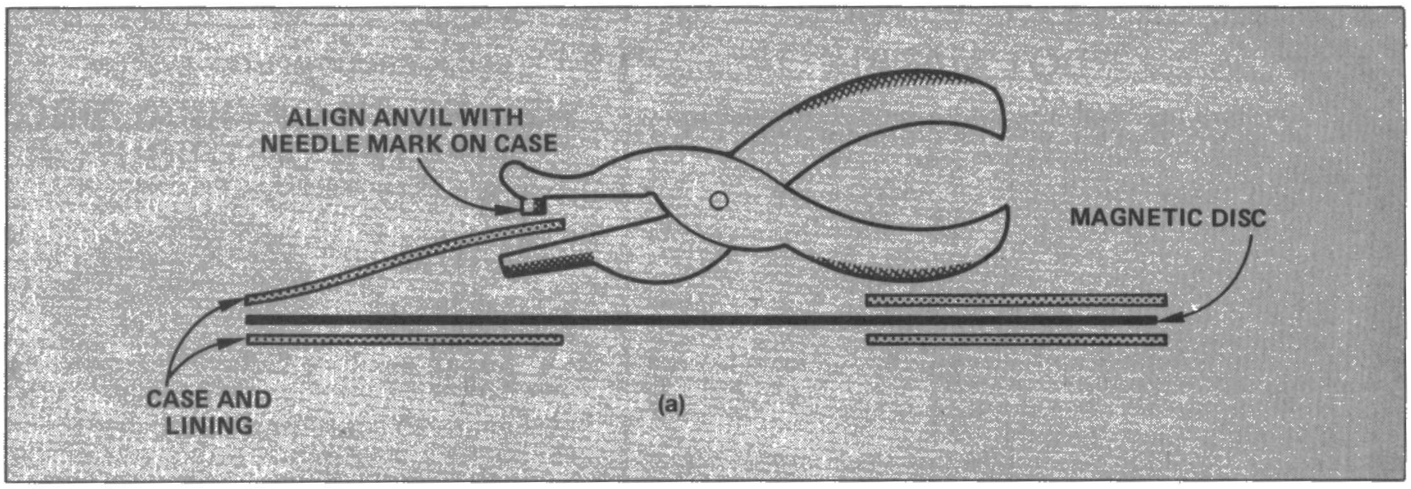
Finally, with the same care, punch out the index hole on the other side of the case, remove all waste material and the disc is ready for use. (You may like to add another label for the second side just to round the job off nicely.)

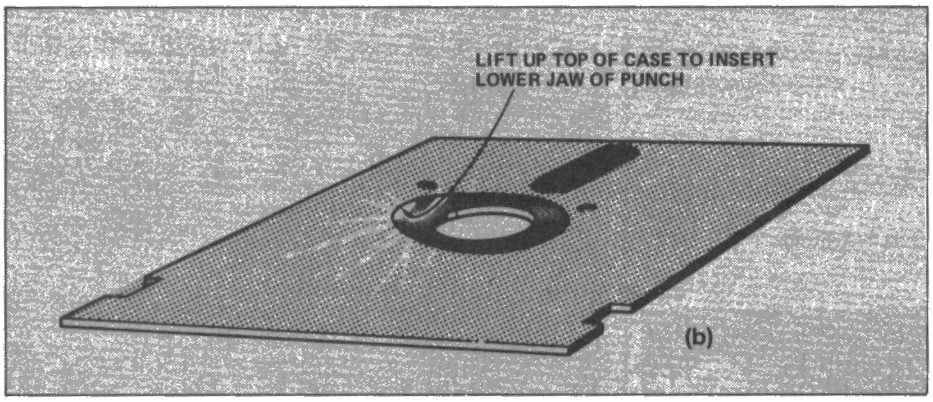
Pros and cons

Some people may still be scared of violating the sanctity of their disc. Apart from the obvious cost advantage, double sided discs are more convenient to use and require less storage space for a given capacity. But naturally the first concern is always with the possibility of damage to the drive followed by concern that disc reliability will be impaired.

An interesting theory argues that as the disc normally rotates in a single direction any debris will be caught by the leading edge of the head access slot. If the direction is then reversed this debris will be carried away from what is then the trailing edge onto the head, causing rapid wear.

In terms of the disc, the main objection is again related to changing the direction of rotation, and that contact with the case lining and the head will polish the magnetic surface in one direction. If the direction of





Figures 5a and 5b. Punching index holes in the case.

rotation is then reversed disc wear would be accelerated. This argument is sometimes accompanied by a suggestion that the case lining is biased in one direction to trap debris and even to polish the disc.

There may be some truth in these theories but it is the *degree* of disc wear that is important, and there are many interrelated factors involved.

Taking the drive first; be assured there is no chance of affecting the mechanics of the drive in any way and the only item which perhaps needs discussion is the read/write head itself.

In normal use debris which mainly consists of oxide from the disc does get on to the head; that is why it needs cleaning from time to time. However if an abnormal amount of debris were released with a reversal of disc rotation, and if it did get on to the head instead of falling harmlessly to the bottom of the drive, surely a large number of data errors would result.

In practice however this does not occur – even when the reverse side of a well used disc is formatted immediately after modification. Neither does the head require cleaning any more often than normal. Therefore it seems unlikely that the head wear theory has any practical consequence especially when you consider that there is not much debris in a disc anyway.

When it comes to the disc, the argument about the bias of the lining is not very convincing; just lift up the case at the drive hole and take a look! Inside you will see that the disc case is lined with a fibrous material, which looks very ordinary and has a totally symmetrical embossed pattern on its surface. As the pattern is not concentric it follows that it will contact the disc surface at every conceivable angle.

Anyone who is interested enough might also like to take a look at the surface of some used standard discs; the variation in surface finish will be quite surprising and in addition, whereas some discs will be quite hard to turn, others will be smooth and turn easily indicating different amounts of friction which presumably leads to different rates of wear.

As if all this is not enough the disc and the case continually flex while being driven, introducing a random variable into the equation.

"I have modified discs of various makes and have about 40 in use, with no signs of any problems so far".

In practical terms it seems that environmental conditions inherent in the floppy disc media itself will affect disc and head life much more than anything introduced by modifying the disc. Also remember you can actually buy drives that are made to accept an inverted disc.

Some practical points

A modified disc can be regarded as two independent discs, each with its own write enable slot, so it is quite possible to have one side protected with a tab stuck over its slot while the other side remains unprotected. Apart from normal use, the second side of a modified disc is handy for storing backup copies of programs. For good security, however, it is best not to back up on the other side of the source disc.

It is a credit to the quality control of manufacturers that – in most cases – both sides of a single-sided disc will format. In fact it is not unusual to find 40 track single-sided discs being used as 80 track double-sided. Don't worry if it turns out that the second side will not format correctly, nothing is lost because you can still use the disc in the normal way, as though it had not been modified.

Before you do any modifications it is wise to first format and verify the normal side a couple of times to ensure that the disc is serviceable, because it is unlikely that you will be able to exchange it after modification; and be warned that the lifetime guarantee carried by certain premium discs will be invalidated.

Finally it is well worth mentioning an aspect that has an overriding effect on disc and head life. Discs and drives do not like damp conditions so make certain you keep both in a warm dry environment and avoid any area where condensation is likely. Although it is not an obvious point, always resist the natural temptation to clear any debris by blowing on a disc, because if a single drop of vapour gets on to the active surface your disc may be written off for good.

The results

This operation of modifying discs proved to be relatively routine and should only take a couple of minutes to carry out. I have modified discs of various makes and have about 40 in use, with no signs of any problems so far.

Pascal in practice

To give our programming Pascal series a practical touch, Adam Denning looks at implementations of the language for two popular micros: the QL and the Beeb.

Here we take a look at two implementations of Pascal for two popular home computers: Acornsoft S-Pascal for the BBC Micro and Computer One Pascal for the QL. It is as well to get the prices sorted out now. S-Pascal costs £24.95 and QL Pascal costs £39.95.

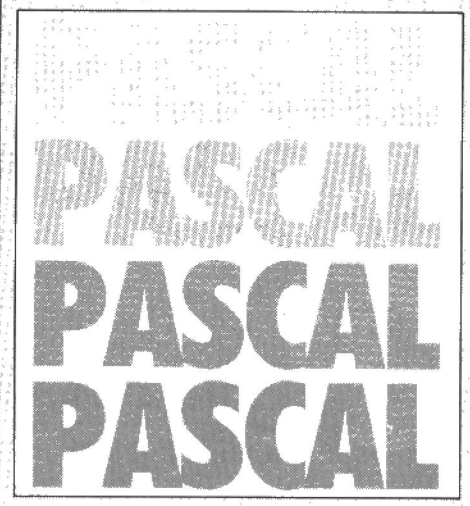
Both come with rather deficient manuals, the S-Pascal documentation in particular seems to take more interest in defining the compiler than describing how to use it. S-Pascal is a teaching subset of full Pascal and therefore cannot handle some of the more complex (and useful!) aspects of the language. It comes on either cassette or disc in a standard Acornsoft package, and has numerous example programs which are all listed in the manual but explained really badly. Considering that the purpose of this compiler is to teach Pascal to the uninitiated this must be seen as a major fault.

The programs range from the cliched Towers of Hanoi puzzle, which is of course recursive, to programs which draw diamonds on the screen. All these programs are well written but the Towers of Hanoi program in particular seems to be an object lesson in showing off. We all know that the BBC Micro is fast, but Acornsoft seems intent on proving it, making the Hanoi program far too fast to see in action. It strikes me that this defeats the object of the exercise. You can also confuse the program by asking for too many discs to be displayed, then the screen just turns into a disorganised mess. Whatever happened to error checking?

The manual also includes full syntax diagrams for the implementation, which seems to be a practice unique to Pascal, but of course it neglects to explain them.

The compiler itself is apparently written in Basic, which is no major problem if it works properly, but it does have some side effects. Programs must be written in lower case or are tokenised by the Basic interpreter, and the raising of PAGE (to fit in the compiler and source code) leaves very little room for programs. If you force yourself to stay in mode 7 then you get the benefit of a full 3 to 4K!

Editing the source is a bit of a pain, using re-directed OSCLI calls which are manifested as new * commands, but compilation is reasonable, invoked by a *COMPILE command. This is probably one of the slowest compilers going, but does have the enormous virtue of showing the 6502 assembly language that it generates on the screen along with the source. As it uses no



Basic ROM routines in the final object code the compiled programs can be saved to cassette or disc and subsequently loaded and run with a *RUN command, but it does also mean that REALs are not supported. (REALs are floating point numbers.) The simple types supported are INTEGER, with the usual 16-bit range of —32768 to 32767, CHAR, which covers 8 bit ASCII characters, and BOOLEAN, which gives the values TRUE and FLASE.

Everything seems to work as one would expect, but this package can only be bought on the understanding that nothing serious can be done with it. This must be severe embarrassment to Acorn as the BBC Micro is promoted as being the 'serious' machine, the darling of intellectual establishments. Yet it still has no serious languages except BCPL.

Compare that to the Spectrum, which now has an almost complete Pascal and a very good C. Acornsoft is due to release the full ISO Pascal as a two ROM system in the late part of this year. It will reputedly cost £125, which seems an awful lot when you consider that no other small micro implementation of Pascal costs anything near this. It also smacks of the second processor system – two years too late.

Computer One's Pascal for the QL, on the other hand, costs £15 more than S-Pascal and yet offers so much more.

It is supplied on Microdrive cartridge with a manual which declares the language and all its resident procedures, functions and pre-defined types. This is far more a system than just a compiler, as it includes a full screen editor: compilation is just one of the options from the menu.

The system is invoked by putting the cartridge in drive 1 and pressing the requisite function key. Curiously, this reserves some resident procedure space and loads a machine code procedure into this area. This procedure is called PASCAL, and in common with all SuperBasic procedures, typing the procedure name invokes it. This then loads the system from drive 1 and results in a menu with eight options shown on the screen. The important options are EDIT, COMPILE and RUN. Each option is selected either by using the cursor keys to move a window down to the option concerned or by pressing the number of the option. Pressing ENTER invokes the option.

"Nothing serious can be done with the S-Pascal package".

The EDIT option loads Computer One's screen editor: not really an ideal type. The Metacomco editor supplied with its assembler, BCPL and LISP, is far superior and in fact this is the one I use to develop Pascal programs. The Pascal editor is very limited in its range of commands, so that the only real options are saving and loading files, moving the cursor keys and inserting text. One interesting thing is that if a program is compiled and results in compile time errors, then these errors are reproduced in the source file as non-deletable lines, which makes the alteration of error prone lines that much easier. When the amended file is re-saved the errors are not saved.

Files which are intended to be source files for the Pascal compiler must end with the **_pas** extension, and the compiler option throws up an error if the specified file is not thus named. The system designates a drive as its default so that in most cases there is no need to type out the file name. If we wanted to compile a file called, say, **mdv2_test_pas** then all we need do is type **test.**

The compiler is loaded from drive whenever it is entered and the compilation process is reasonably fast. This stage does not compete with, say, Hisoft Pascal, but it is doing a drive to drive compilation. A screen listing is optional. The object code is produced as a file with the same main name as the source file but with the extension **_qlp,** which we are told stands for 'QL Pascal'. Fair enough.

The object code is *not* 68000 machine code, but an interpretive code similar to the CINTCODE system used by many small BCPL systems. While an interpretive code is going to be slower than its pure machine code equivalent, it does have a number of advantages. It is smaller, for a start, and the choice of a suitable interpretive code makes the compiler a lot easier to write and a lot more efficient than an optimising machine code compiler. The compiler is of course written in Pascal, but unfortunately the source is not reproduced!

Computer One Pascal takes advantage of QDOS and the floating point routines therein, so its range for REALs is the same as SuperBasic's. But this is rather unfortunate: despite the fact that the floating point range is enormous (1e-615 to 1e615), the

PASCAL
PASCAL
PASCAL

accuracy is less than is normally desired, being 7 significant digits. Its integer range is superior to SuperBasic's, taking the full 32 bit size of -2^{31} to $2^{31}-1$.

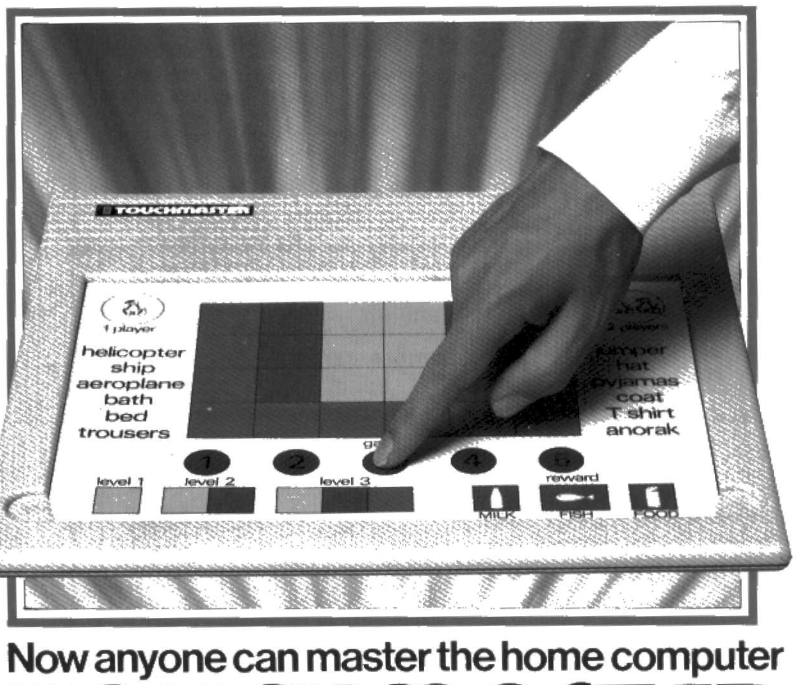
The compiler comes with many in-built functions and procedures to take advantage of the QL's input, output and screen. All SuperBasic routines to draw complicated graphics are available in Pascal with the same names and parameter structure, and there are even defined types to ensure that certain parameters – such as colours – are valid and easy to use.

I did find problems with the READ procedure, which is supposed to assign values read in from an output device to variables. It seems that the procedure will only accept numbers which are actually of the type specified by the parameters. There appears to be no type coercion, so that specifying a real variable as a READ parameter seems only to work if there is a number that is *definitely* real (ie 34.5 rather than 34).

The compiler comes with a Basic clone program to enable easy backup, and there are four Pascal examples provided. Two of these are becoming so common as example programs that they are terminally boring – Hanoi and the Fibonnacci series! The other examples are a program to draw a globe and a very interesting program which talks to QDOS directly and enables the fast backup of Microdrive cartridges. The source code of this shows how easy it is to access QDOS from Pascal, but as it says in the comments, it is unlikely to make much sense unless you have the QDOS documentation. Basically, it reads the directory of the source drive and transfers each file across to the destination drive. This is harder to do but more efficient in machine code, as we showed in the last issue.

There is no doubt that this compiler is a superior product to Acornsoft's Pascal, admittedly helped by the more advanced features of the QL, but it may be worth waiting a while to see what the Sinclair-branded product is like. Going on the experience of the assembler it is likely to be less useful than this product.

IT TRANSFORMS THE HOME COMPUTER OUT OF ALL RECOGNITION



Because Touchmaster is a touch sensitive surface which effectively bypasses the keyboard, it has none of the keyboard's complications, typing skill requirements or potential errors.

To operate Touchmaster, you simply slide an overlay onto its surface, load the matching Touchware into your computer and touch the overlay.

For repeatability and resolution, no other ostensibly comparable pads can touch Touchmaster.

In fact, Touchmaster's unique technology makes it state of the art when it comes to such pads.

Other pads might fairly be described as peripherals. Touchmaster goes a lot further: it respecifies the home computer.

Now anyone can master the home computer

TOUCHMASTER

For full details, contact Touchmaster Limited, PO Box 3, Port Talbot, West Glamorgan SA13 1WH, or phone Teledata (01) 200 0200

Parlez-vous PASCAL

Gerry Davies continues his series on everything you need to know about Pascal, this month with a look at user defined types and arrays.

In the first part of this series we introduced the concept of types. To recap, all variables in a Pascal program must be declared, together with their type. These variables can then only hold values of the same type as they were declared to be. We saw the two basic numeric types, REAL and INTEGER, as well as some control structures using them. In this part we will consider the other two basic types, namely BOOLEAN and CHAR and how the control structures can use these types. Further control structures will be described and the basic concepts of user defined types discussed.

Boolean variables

A Boolean type is one that can hold a truth value, either TRUE or FALSE, and no other value. (These values are reserved words in Pascal.) In the first part we looked at REPEAT and WHILE loops. If you remember, each of these had a conditional statement whose value determined the action of the loop. This conditional statement is more correctly called a Boolean expression as it can only take the two truth values. A Boolean variable can therefore be used to hold the result of a Boolean expression.

You can now see that the loop statements are simply producing a Boolean value and acting accordingly, so we could do this explicitly. Given a declaration of 'Finished' as Boolean – declared in the same way as other variables – we can write:

```
REPEAT
  READ (Value) ;
  Sum := Sum + Value ;
  Finished := Value = 0 ;
UNTIL Finished ;
```

Notice the strange-looking assignment to 'Finished'. Remember that a single equals sign is a relational operator – it compares the two values on either side of it. This assignment therefore sets 'Finished' TRUE if 'Value' is zero, otherwise 'Finished' is TRUE. See how similar the English description of a Pascal program section is to the program itself! It is this that makes Pascal programs so easy to read and understand.

The above loop will perform exactly the same job as that presented last month:

```
REPEAT
  READ (Value) ;
  Sum := Sum + Value ;
UNTIL Value = 0 ;
```

You can see how sensible use of indentation makes the loop structure clear. The two inner statements are obviously those controlled by the loop, just from its layout on the page.

The IF statement

A further control statement, IF, allows the choice between sections of a program to be executed, depending on the value of a variable. It is therefore very similar to an IF in a BASIC program. Again the controlling statement is a Boolean expression or variable. There are two versions of the IF statement in Pascal. The first has the familiar form

```
IF condition
  THEN statement :
```

This means exactly what it says. If the condition (a Boolean expression) evaluates to TRUE then execute the statement. Control then passes to the next part of the program, so if the expression was FALSE, the controlled statement is not obeyed. For example, if we wanted to ensure an integer was positive, we could negate it if it were negative:

```
IF Value < 0
  THEN Value := –Value ;
```

Thus if 'Value' is less than zero – so negative – the condition is TRUE and the statement will negate 'Value'.

The other form allows selection between two alternative actions:

```
IF condition
  THEN statement1
  ELSE statement2 ;
```

Two things are important here. First notice that there is no semi-colon at the end of 'statement1' but there is one at the end of the second statement. This may seem a trifle odd at first, but you must remember that the semi-colon signifies the end of a complete statement. The IF statement as a whole must therefore be terminated with a semi-colon, but its constituent parts must not. It is the ELSE that lets the compiler know that 'statement1' has finished.

Secondly, the operation of the IF statement is as you might expect. If the condition is TRUE then 'statement1' is executed, else statement2 is executed. Under no circumstances can both 'statement1' and 'statement2' be executed.

"See how similar the English description of a Pascal program section is to the program itself".

As before, these controlled statements can be compound if desired. Obviously when a BEGIN END bracket is put around a group of statements to form a compound statement, each inner statement must be terminated with a semi-colon. For example, suppose we want to write out the fact that we needed to negate our integer above:

```
IF Value < 0
  THEN
    BEGIN
      Value := –Value ;
      WRITELN ('Value was negative') ;
    END
  ELSE WRITELN ('Value was positive') ;
```

This shows the correct use of semi-colons, and the use of compound statements within an IF statement.

The controlled statements can be any valid Pascal statement, as with all other constructs. In particular, it could be another IF statement. This allows you to perform multi-way selection by building a nest of IF statements, such as

```
IF condition1 THEN statement1
ELSE IF condition2 THEN statement2
ELSE IF condition3 THEN statement3
ELSE statement4 ;
```

The last ELSE will always apply to the most recent IF. Therefore 'statement4' is only executed if all of the conditions are FALSE.

Character variables

The basic type CHAR is used to hold single characters. There is no basic type for holding character strings (or just strings as they are often known). The Pascal techniques for string handling will be described later on. A CHAR variable is declared in the same way as the previous types, for example:

```
VAR
  A, Character : CHAR ;
```

declares the variables 'Character' and 'A' to be of type CHAR.

As you might expect, we can READ and WRITE CHAR values, as well as performing assignments. There is also a limited facility for type conversion from numbers to characters and vice versa. In order to assign characters, we must have a technique of distinguishing the character literal from the surrounding program text. This is achieved by putting single quote marks around the literal. Thus to assign the letter A to our variable, we must enter:

```
Character :+ 'A' ;
```

If these quote marks were left out, we would assign the value held in the variable 'A' rather than the letter 'A' itself. In order to help avoid confusion, it is good practice to use meaningful variable names, such as 'Character' rather than just 'A'. Only those characters present on a keyboard – the printable characters – can appear as literals.

Obviously, there are no arithmetic operations to handle characters. After all, what do you expect the result to be if you add two characters? You can however compare characters using the relational operators introduced last time. These assume alphabetic order, so 'A' is less than 'B' and so on. But you cannot assume that numbers and letters will be in a given order; this depends on the internal representation and could be different for different computers.

There are also two type conversion operations available: ORD and CHR. The first converts characters to the bit pattern they are represented by. This also depends on the computer the program is running on, but usually the ASCII code is used. Thus

```
Value := ORD('A') ;
```

would assign the integer 'Value' to be 65, as the ASCII code for 'A' is 65. The reverse

"A Boolean type holds a truth value: true or false".

operation is performed by CHR. Thus to send a carriage return character to the OUTPUT device; you can explicitly send it with a

```
WRITE( CHR(13) ) ;
```

as the ASCII for a carriage return is 13. This operation also allows you to overcome the fact that a carriage return is not a valid literal. So

```
VAR
  Return : CHAR ;
BEGIN
  Return := CHR(13) ;
```

assigns the carriage return byte to the variable 'Return'.

Character constants are also permitted, using the same notation as before. They can help to produce easily read program sections. For instance, suppose a program needed to ask its user if they wanted instructions. This can be done by prompting for an answer and seeing if the reply is yes or no:

```
  Yes = 'Y' ;
  NO = 'N' ;
VAR
  Answer : CHAR ;
BEGIN
  WRITELN ('Do you want
             instructions ?') ;
  READLN ( Answer ) ;
  If answer = Yes
    THEN write out the instructions . . .
```

Because the READLN will read the first character the user types, this code will always work even if the answer 'Yes' is given in full. Notice that the condition for the IF statement is a character comparison, using the constant value declared for 'Yes'. A value is also declared for 'No' as this may be needed elsewhere.

Comments

A further aid to program readability is the comment. In BASIC you can usually use a REM statement to add a comment. Pascal allows the inclusion of comments anywhere in a program, even between words if necessary (but not in the middle of a word please!). Comments can be as long as you like, and can contain anything except another comment. Obviously, there must be a mechanism to let the compiler know it has encountered a comment, and to ignore it. This mechanism is the curly bracket pair '{}'. Any text contained between these brackets is ignored by the compiler. Some keyboards do not have curly brackets on them, so an alternative form is also allowed, '(*' to start a comment and '*)' to end it.

Comments are particularly useful at the start of a compound statement; for instance, as a reminder of what the statement will do. In the example above we said we would write out the instructions. This could be done in a compound statement:

```
If Answer = Yes
  THEN
     BEGIN {write out the instructions}
     END ;
```

The above code fragment is a valid Pascal statement, and will compile and run, given the necessary surrounding declarations and program body. Notice that the compound statement doesn't even contain any statements! We needn't put these in until we know what they are. Pascal treats the statement as a perfectly valid null statement which will have no effect. This allows you to build up programs in small, easily tested stages. Once we have got the main program working, we can add the details such as code to write out the instructions.

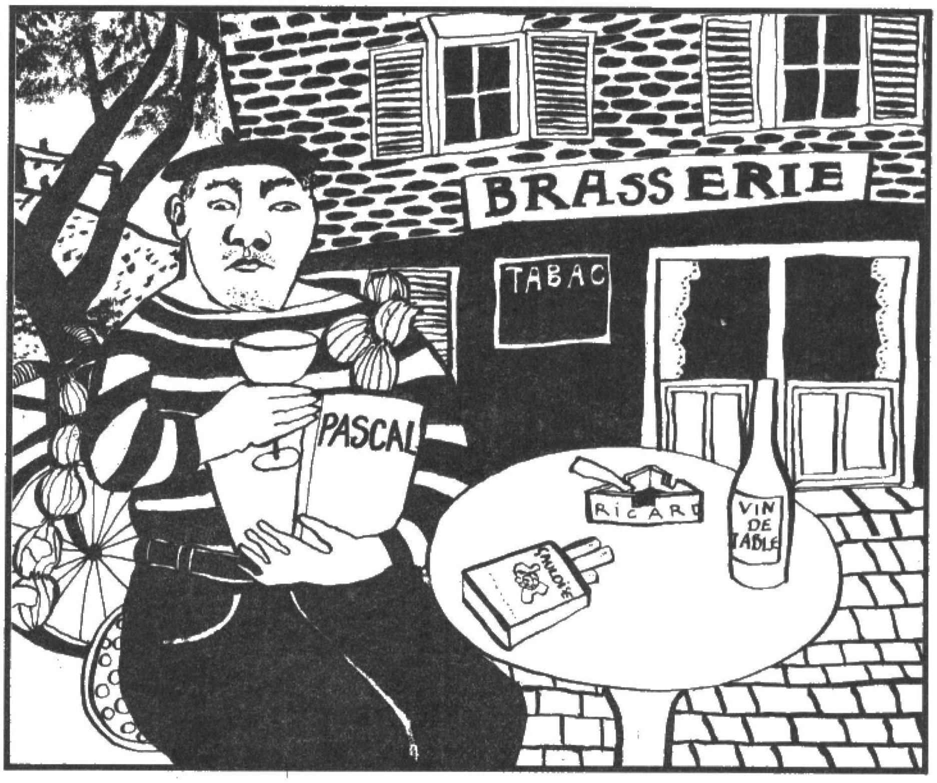
A further use for comments is in testing. As a comment bracket pair can appear around anything, we can put it around Pascal statements. These statements will therefore become comments, so will be ignored. You can therefore 'comment out' sections of program that don't seem to work correctly to investigate the bugs in other parts of your program.

User defined types

The main power of Pascal over most other languages comes from its ability to allow you to define your own types. We have now seen all four of the basic pre-declared types, but can add to these at will. In the declaration section of a program, we have so far seen the CONST definitions, followed by the VAR declarations. In between these can appear any number of TYPE definitions. We shall now look at some simple user defined types, and see why they are so powerful.

Enumerated types

The simplest user defined type is the enumerated type. We often need to manipulate values other than numbers, for instance months in a year, days of the week and so on. In Pascal, we can declare a type 'Months' to contain the months of the year:



```
TYPE
  Months = (Jan, Feb, Mar, Apr, May,
    Jun, Jul, Aug, Sep, Oct, Nov, Dec) ;
```

We can now declare a variable to be of this type, and use it in the normal way, so

```
VAR
  Month : Months ;
```

What does this mean? We have said before that the type of a variable defines what values it can take. Thus a variable of Boolean type can only hold one of two values, TRUE or FALSE. Our variable 'Month' can similarly take any of the twelve values 'Jan', 'Feb' and so on! It is important to distinguish between a variable and its type. The declaration of the TYPE 'Months' above only tells the compiler that variables can be declared to be of this new type. It is not until we actually declare a variable to be of this type that such a variable will exist.

So what can we do with this variable? Well, anything you might want to, such as compare it, assign it and so on. For instance

```
IF Month = May
  THEN WRITE ( 'Spring is here . . .' ) ;
```

You can also use the variable in a FOR loop. Suppose we want a program to read in the rainfall for each month and compute the average rainfall in a year:

"You can 'comment on' sections of a program that don't seem to work correctly".

```
PROGRAM Rain ( INPUT, OUTPUT ) ;
{Program to compute average rainfall
over the year}
  TYPE
    Months = (Jan, Feb, Mar, Apr, May,
              Jun, Jul, Aug, Sep, Oct,
              Nov, Dec) ;
  VAR
    Month : Months ;
    Rainfall, Average, Sum : REAL ;
  BEGIN
    Sum := 0 ;
    FOR Month := Jan TO Dec DO
    BEGIN
      READLN (Rainfall) ;
      Sum := Sum + Rainfall ;
    END ;
    Average := Sum / 12 ;
    WRITELN ( 'Average monthly
              rainfall was ', Average ) ;
END.
```

Notice the use of a comment to remind you what the program does, together with sensible names for the variables and types. We shall see more uses for enumerated types later on, but hopefully their power is already clear.

Sub-range types

Often we want certain variables to hold only a part of a full range of values: for example a variable 'Summer' that can only take values of summer months. This is easily achieved. We can declare a type 'SummerMonths' to take a sub-range of the full range of valid months:

```
TYPE
  Months = (Jan, Feb, Mar, Apr, May,
            Jun, Jul, Aug, Sep, Oct,
            Nov, Dec) ;
  SummerMonths = Jun . . Aug ;
```

```
VAR
  Month : Months ;
  Summer : Summermonths ;
```

The special symbol '. .' says take those values from 'Jun' to 'Aug'. Thus our variable 'Summer' can only take the values 'Jun', 'Jul' or 'Aug'. It is still considered to be of type 'Months', but can only hold a sub-range of the values in that type. Thus an assignment

```
Month := Summer ;
```

is valid, but the assignment

```
Summer := Dec ;
```

would be rejected by the compiler. What happens with assignments of the form

```
Summer := Month ;
```

depends on the value of 'Month' at the time of the assignment. If it falls within the sub-range, the assignment is allowed, otherwise an error has occured. This should be checked by the program as it is run, but some compilers do not bother to produce the necessary code!

You can declare sub-ranges of the pre-declared types INTEGER and CHAR, for example:

```
TYPE
  Letter = 'A'. .'Z' ;
  Digit = '0'. .'9' ;
```

Sub-ranges of REALS are not allowed, and some compilers place a limit on the number of values a sub-range can have. Clearly a sub-range of the Boolean type is not possible, as it only contains two possible values.

Whilst the same job can be done without the enumerated type and sub-range facility – by declaring constants – it is not as clear or easy to use. As the type of each variable is checked, the compiler ensures that types are not mixed in assignments, and so reduces errors. If you have declared the type 'Months' as above, and also declare the following enumerated type and variables:

```
TYPE
  Days = (Mon, Tue, Wed, Thu, Fri, Sat,
          Sun) :
VAR
  Day : Days ;
  Month : Months ;
```

Then an assignment of the form

```
Month := Day ;
```

will be rejected by the compiler because 'Month' and 'Day' are not of the same type.

"The main power of Pascal over most other languages is its ability to allow you to define your own types".

The array – a structured data type

The concept of the array should be familiar to you. BASIC supports a rather crude form of array, as do most other languages. They all have the same basic form: a collection of data (known here as the elements of the array) and a subscript or index giving access to the individual elements comprising the array.

In Pascal, the two parts of the array are very distinct. The array elements can be of any type, including user defined types and the basic types. All elements in a given array have to be of the same type. The index must be of an ordinal type. That is, it must be of a type that can be represented with a single integer. Ordinal types therefore include CHARs, INTEGERs, BOOLEANs and enumerated types – not much of a restriction! REALs are not allowed as indices.

Suppose we want to build a table of rainfall over the year. We can use some of the types left over from our previous examples, together with a new type to hold the table:

```
TYPE
  Months = (Jan, Feb, Mar, Apr, May,
            Jun, Jul, Aug, Sep, Oct,
            Nov, Dec) ;
  Table = ARRAY [Jan. .Dec] OF REAL ;
VAR
  Year : Table ;
  Month : Months ;
```

This declares a new type, called 'Table'. It is an array of reals, containing one REAL element for each of the twelve months. The sub-range operator '. .' is used to signify the range of valid indices – from 'Jan' to 'Dec'. 'Year' is therefore a table with twelve entries, which can be accessed individually with a variable of type 'Month'. Don't forget to declare a variable to use as an indix – here it is called 'Month'. Some keyboards do not have the square bracket pair '[]' on them, but an alternative form is also allowed: '(.' for the open bracket, and '.)' for the closed bracket.

How can we access the elements in this array? By using an index, as already mentioned. So

```
Year[Mar]
```

refers to the third element, the entry for March. As the elements are REAL, they can be manipulated in the same way as other REAL variables. So if we wanted to work out the average rainfall, with all the figures already saved in our table, we could type:

```
Sum := 0 ;
FOR Month := Jan TO Dec DO
  Sum := Sum + Year[Month] ;
Average := Sum / 12 ;
```

where 'Sum' and 'Average' are both REALs. At each pass through the loop the variable 'Month' takes the next value in the type. Therefore each element of the array 'Year' is accessed in turn.

Array elements can be on either side of an assignment – otherwise they could never get values in the first place. They can also be read and written, but only individually. If you want to output all of the elements in an array, you must use a FOR loop and write each one in turn.

You can also assign an entire array in one statement. If 'Year1' and 'Year2' are both of the same type, 'Table', then this assignment:

```
Year1 = Year2 ;
```

will copy each element of the 'Year2' array into the corresponding element in 'Year1'. Unfortunately, you cannot compare entire arrays in one statement, this must be done one element at a time. In fact, no arithmetic operations are allowed on entire arrays, only on individual elements.

One problem with Pascal arrays is that they can only have one dimension. However, that is a very pedantic way of looking at the facts. An array element can be of any type, including a further array! This means we can build arrays of arrays – in effect a multi-dimensioned array. It also means that there is no limit on the number of dimensions an array can have, though certain compilers do put an arbitary limit on this.

Suppose we want to extend our rainfall table to cover the last ten years. We could do this by declaring ten separate tables, one for each year. Alternatively we could build a two dimensional array: an array of year tables:

```
TYPE
  Months = (Jan, Feb, Mar, Apr, May,
            Jun, Jul, Aug, Sep, Oct,
            Nov, Dec) ;
  Table = ARRAY [Jan. .Dec] OF REAL ;
  BisTable = ARRAY [1974. .1984] OF
             Table ;
VAR
  Rainfall : BisTable ;
```

The type 'BisTable' is such an array. Notice we have used an integer sub-range as the index type. This enables us to refer to individual years with the actual year number. How do we now access this array? If we want to find the rainfall in December 1976, we must type:

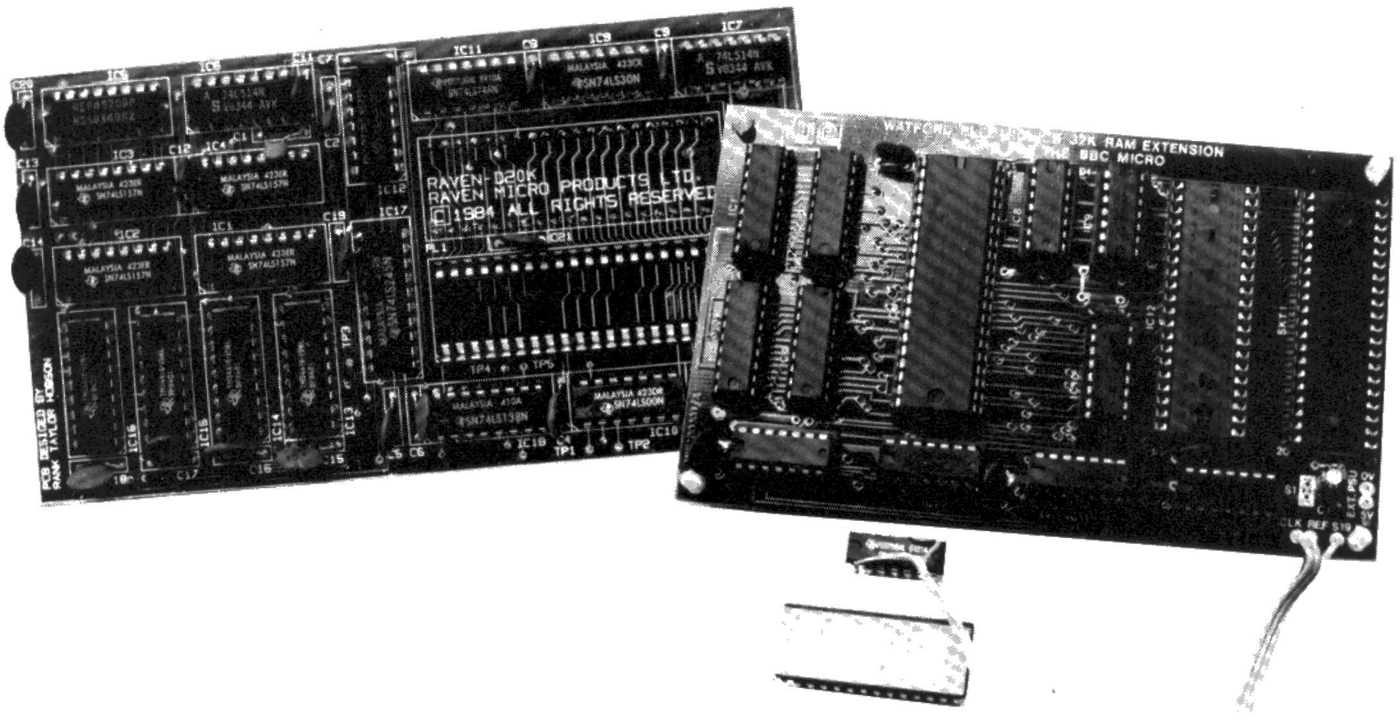
```
Rainfall[1976][Dec]
```

The first index, '1976' selects the correct array of type 'Table' from which we can read the required months figures. The second index 'Dec' actually refers to the required entry in the array for 1976. There is a shorthand notation for this type of access, missing out the inner brackets:

```
Rainfall[1976,Dec]
```

has the same effect.

In this part we have introduced user defined types, and arrays. In the next part we will extend user defined types to include a further powerful data structure. We will also look at the one remaining control structure, and subroutines which can be used to split your program into small, manageable blocks of code.



BBC MEMORY EXPANSION

Tony Jones compares two competing memory expansion systems for the BBC micro. Both enable programs of up to 28K to be run regardless of screen mode.

When it was initially launched, the provision of 32K of RAM within the BBC computer seemed generous when compared to other home micros available at the time. Today though, 64K of RAM is rapidly becoming the norm and the Beeb seems rather short on memeory when compared to more recent designs and this memory facility can all too often be a problem, particularly if any high res graphics displays are required within a program. In this case the amount of user RAM rapidly diminishes to the point at which the computer hardly gives a ZX81 a run for its money. A number of solutions to this problem have found their way onto the market during the course of this year and this review will be looking at two of them.

Readers of *E&CM* can hardly have failed to note that there has been a certain amount of legal dispute surrounding the design of BBC memory expansion boards. Allegations involving patent infringement and plagiarism etc have been thick on the ground in recent months. This review will not concern itself with these, although elsewhere in this issue you will find the latest news on this aspect of the story.

▷

Raven Expansion Board

*RON	Enables the board.
*RVFF	Completely disables the Raven board.
*RVTEST	Allows correct operation of the system to be confirmed.
*RSTAT	Returns the status of the board (on/off).
*SSAVE	Automatically saves the current screen to tape or disk.
*SLOAD	Loads a saved screen to the appropriate location.
*RZAP	Enables the user to examine memory contents in a similar way to Disk Doctor's MZAP routine.
*ROZAP	Enables the user to edit sections on a disk similar to Disk Doctor's DZAP routine.
*RAM	Allows the user to load a file into a specified 'sideways RAM' slot.
*FRON	Alternative to *RON command that enables the board in a different fashion.
*RPAGE	Similar to *FRON but allows the user to define the area in which the system vectors will be located.
*HELP RA	Displays the current values of PAGE and HIMEM.

Watford Memory Expansion System

*RAMON	This command turns on the RAM board to free the memory normally used by the screen.
*RAMOFF	This command turns the RAM board off.
*CLAIMON	Releases for the user the 256 byte workspace used by the system when BREAK is pressed.
*CLAIMOFF	Frees the 256 byte workspace.
*BUFFERON	Turns the buffer on and can have two suffixes S/L and a numberic 0-8. The S suffix selects the 12K buffer option, the L suffix the 32K buffer. The numeric suffix allows the additional buffer RAM to be assigned to any of BBC micro's internal buffers. The default selection is for a 12K printer buffer.
*BUFFEROFF	Turns off the buffer.
*PURGEOFF	Prevents the large buffer being purged on ESCAPE or *FX15 etc. which would be undesirable in some cases when working with a large buffer.
*PURGEON	Enables purging of the buffer.
*HELP BUFFERON	To obtain status information.
*HELP RAMSTATUS	on the current status of the system.

Friendly bank manager

Very crudely, the BBC micro divides the 64K addresses that an 8-bit micro is capable of addressing into two areas. One of these is occupied by the ROM containing the computer's operating system and the BASIC interpreter, the 32K of RAM being allocated to the second area of the memory map. From this it is evident that, if some extra RAM is going to be added to the system, some form of bank switching of memory will be required.

In order to achieve the necessary switching of memory without wholesale alterations to the pcb of the BBC micro both the boards reviewed here have adopted a similar technique. This involves removing the computer's 6502 processor and replacing it with a small pcb containing the additional memory and the control electronics. These add-on boards also have a blank socket in which the 6502 is inserted. Control of the system is the job of EPROM-based software that is mounted in a vacant sideways ROM socket.

For a more detailed description of the operation of such systems it is suggested that readers refer to the first part of our Memex project published in April of this year (note however that we were unable to complete the publication of the project as we ourselves were caught up in the legal difficulties referred to above).

Summarising specs

At a glance the specifications of the Watford and Raven memory expansion boards are rather similar. This is hardly surprising given the limited design options available to anyone wishing to incorporate extra RAM within the BBC in such a way that its operation will, in the main, be transparent to the user. Both systems allow programs of up to 28K to run regardless of the selected screen mode. Both make some additional OS calls available and both are installed in a similar fashion. The Watford board offers the attraction of a printer buffer though, while the Raven board manages a more extensive range of additional OS calls.

Users who can handle the installation of a sideways ROM IC without the aid of a dealer will have no problems in fitting either expansion system to their micro. In both cases it is necessary to fit an EPROM which holds the controlling software in a vacant sideways ROM socket and to remove the Beeb's 6502 MPU. In the case of the Raven system, installation is completed by inserting the 6502 processor in the pcb that holds the additional memory and inserting this assembly into the vacated socket on the BBC micro's board.

The Watford system adopts a slightly different approach. After removing the computer's processor to a socket on the expansion pcb a ribbon cable is used to connect the micro to the additional memory (this allows more flexibility in the sighting of the additional pcb). It is also necessary to replace IC86 of the BBC with one supplied with the Watford board and to make two flying connections to this new IC and to link S19 on the micro's main pcb.

The manuals supplied with both systems take the user through the installation process in a clear, easy-to-follow fashion and in both cases the conversion will only take minutes.

As a major aim of both products is to provide a system that is transparent in operation it is difficult to comment on the performance of the two systems, save that they both meet the required objective of the provision of extra RAM. Instead of describing the operation of the systems, the best way to assess the capabilities of the systems is to study the list of commands that are used to control the operation of the boards.

Both systems are well made and the choice between them will be a difficult one for many users. The ability of the Watford system to be configured as either a 12 or 32K printer buffer will obviously be a major attraction for many users. There are others though that will value the comprehensive range of additional utilities that are built into the firmware of the Raven board. In the end it will depend on which of the rival systems unique qualities you rate most highly.

NO HOME COMPUTER IS COMPLETE WITHOUT IT

Because Touchmaster opens up the home computer.

First, to the full range of users, even the very young.

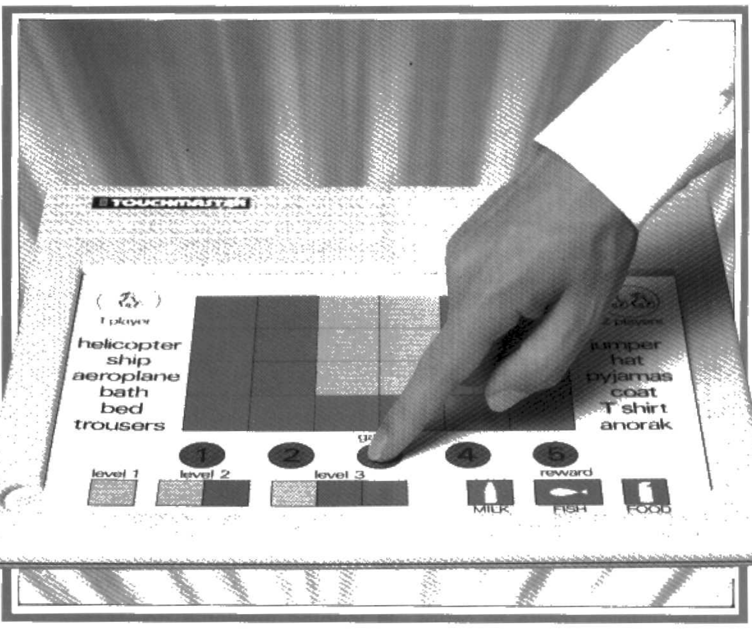
Secondly, to a whole new style of software ("Touchware"), which is not limited by the keyboard.

Touchware goes all the way from early education packages to sophisticated graphic programs.

What's more, comparable pads are limited because their surface is composed of either a bank of switches of finite size or carbon tracks laid in a grid.

Or they need a delicate electronic stylus. Or they offer a limited working surface. Touchmaster's full A4 printed surface, by comparison, is totally linear.

It has a resolution of 256 x 256 points (that's what makes Touchmaster so flexible).



Now anyone can master the home computer

TOUCHMASTER

For full details, contact Touchmaster Limited, PO Box 3, Port Talbot, West Glamorgan SA13 1WH, or phone Teledata (01) 200 0200

FREE EVERY MONTH

your ROBOT

BRITAIN'S FIRST ROBOTICS MAGAZINE

Each month, Your Robot reports on the latest news from the world of low cost robotics. In addition in-depth features explain the theory behind the operation of robots and computer control systems.

Degree course for roboticists?

The Open University has recognised the urgent need to explain to engineers, and indeed the general public, the realities of today's robots. The BBC Open University production centre have just completed work on a series of five 25 minute programmes which between them provide a comprehensive overview of the current applications of a variety of robot systems.

Transmission of the programme begins at 10.35 on Saturday December 1st. This is a scene-setting programme with the title "Have the robots arrived?" The programme shows the extent to which robots have penetrated the manufacturing industry and features an interview with Joe Engleberger, the "father of robots".

Programme 2 looks at the operation and programming of a number of different robots while the third programme examines the way in which sensors can increase the versatility of a robot by providing information about its working environment.

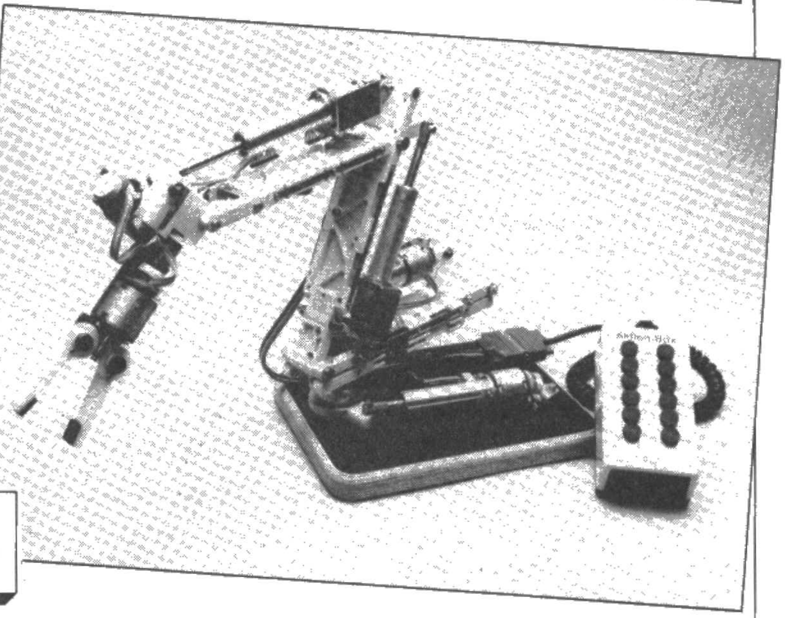
The penultimate programme traces the use of robots in the glamour area of robotics and assembly and includes an interview with the head of the design group responsible for the PUMA arm. The series concludes with a programme that looks at the implementation of robot systems with an in-depth look at the way in which Austin Rover have made use of robots in their manufacturing operation.

The Open University have a reputation of producing interesting programmes that present a great deal of information in a lively and interesting fashion. Anyone interested in robotics should find much of interest in the series.

Transmission times

Have robots arrived? – 1st December 1984.
Making the right moves – 8th December 1984.
Making sense of uncertainty – 15th December 1984.
Assembly: Suitable for robots? – 5th January 1985.
Implementing robot systems – 12th January 1985.

STARTS NEXT MONTH! NEW SERIES Build your own Remcom arm



'Your Robot' News Desk is on 01-833 0846

Teaching Tomys to walk

Tomy's voice recognition unit, going under the name of Verbot, should be one of the biggest selling robots over the Christmas season. This prediction is based on the fact that the robot is one of an increasing number of products that are more toys than true robots. Verbot can be trained to recognise eight verbal commands that control its movement forwards, backwards, rotate it left and right, allow it to lift up and put down objects, to flash its 'eyes' and to terminate the current operation.

Most of the above actions are self-explanatory but the operation of Verbot's arms is rather unusual. The two arms are mounted on Verbot's shoulders and at rest lie flush with its body. When commanded to do so, the arms lift up to a prayer-like position in front of the robot. Any object in front of Verbot at the time is thus picked up. A reasonably effective system but objects must be restricted to small thin items; paper probably being the easiest of things for Verbot to pick up.

The robot is supplied with a 'wireless' microphone into which commands are uttered. When first switched on each command is spoken into this microphone while one of eight buttons on the front of the robot is pressed. The manual accompanying Verbot suggests that commands should be of between ½ and 1½ seconds in duration and that each should sound 'very different'. Thus rather than attempting to train the robot to recognise "turn right" and "turn left", one of the commands should be altered, using the word "rotate" rather than "turn". A LED on the robot indicates that a word has been successfully stored in memory.

In operation Verbot proved surprisingly reliable considering its low cost.

Most commands were recognised at the first attempt and the robot could be trained to respond to a wide range of voices although it performed better when 'listening' to the higher pitch of a female voice rather than deeper masculine tones. One unfortunate aspect of the design is that the robot 'loses' its memory every time it is switched off and must be retrained when it is next used. To train it to recognise all eight commands takes some five minutes and some form of battery back up for the memory would have been a welcome feature.

One minor point to mention is the fact that the wireless link used by Verbot operates at a frequency of 49.860MHz. This is a part of the radio frequency spectrum that is set aside as a mysterious section of the BBC system and is definitely not for use with toy robots. The power of the transmitter is, however, so low (range is only a matter of feet) that use of Verbot is unlikely to interfere with any other transmissions. There is little likelihood of Men from the Ministry bearing down on Verbot owners putting the robot through its paces!

Verbot is definitely fun to operate and while not capable of serious use does demonstrate some important aspects of the general operation of robot systems.

Verbot will be on sale in many toy shops and department stores over the Christmas period and will undoubtedly be a crowd pulling performer.

NEWS

Your Robot, 155 Farringdon Road, London EC1R 3AD. **Editor:** Gary Evans, **Deputy Editor:** William Owen, **Production Editor:** Liz Gregory, **Advertisement Manager:** Richard Jansz, **Advertisement Production:** Yvonne Moyser.

BIGGER BUGGY BUILDING

This month Richard Sargeant shows how to add sensors and an extra interface board to the Hardy Buggy featured in last month's issue.

At first sight the usefulness of the 7 or 8 bit Centronics Port appears to be fading fast as more and more information needs to be passed between the computer and the robot. To recap, we are already using 4 outputs for the driving motors, 1 output for the pen lift and one output for the third wheel arrangement. We could use B6 and B7 to select 1-of-4 inputs to be read by the Centronics BUSY line, but that is no real help since we're reading 2 inputs already (from pen lift and the third wheel), so two bump sensors would be the limit. The question is, should we replace the Centronics connection? – to which the answer is, not yet.

MULTIPLEXED I/O

Figure 1 shows a possible solution to the problem of multiplicity of lines. The Centronics or other parallel port is retained to control two 8-stage shift registers. These are CMOS devices, but if you obtain the buffered variety (suffix B) then they will drive 1 standard TTL gates or a couple of LS TTL gates without any trouble. Note the 2K2 resistors which assist conversion from the TTL parallel port to the CMOS inputs.

The 4049B is an 8-stage shift-and-store register which has data shifted on positive transitions of the shift clock. The data in each shift register stage is transferred to the storage register when the strobe input is high. Data in the storage register appears at the outputs whenever the output enable is high, but otherwise the output are open circuit.

The 4021B is an equally versatile package which can be used as a simple 8-bit-parallel to 1-bit-serial shift register.

The software needed to control the two shift-registers needs to be in machine code – a set of subroutines in fact. One such routine might, for example, take a single byte, present each bit in turn at D0, create a clock pulse on D2 and then, after 8 bits had been transferred, put a high out on D1 so that the bits in each stage of the register could pass as 1 byte to the buggy. There is quite a lot of bit manipulation required along the way. A similar routine

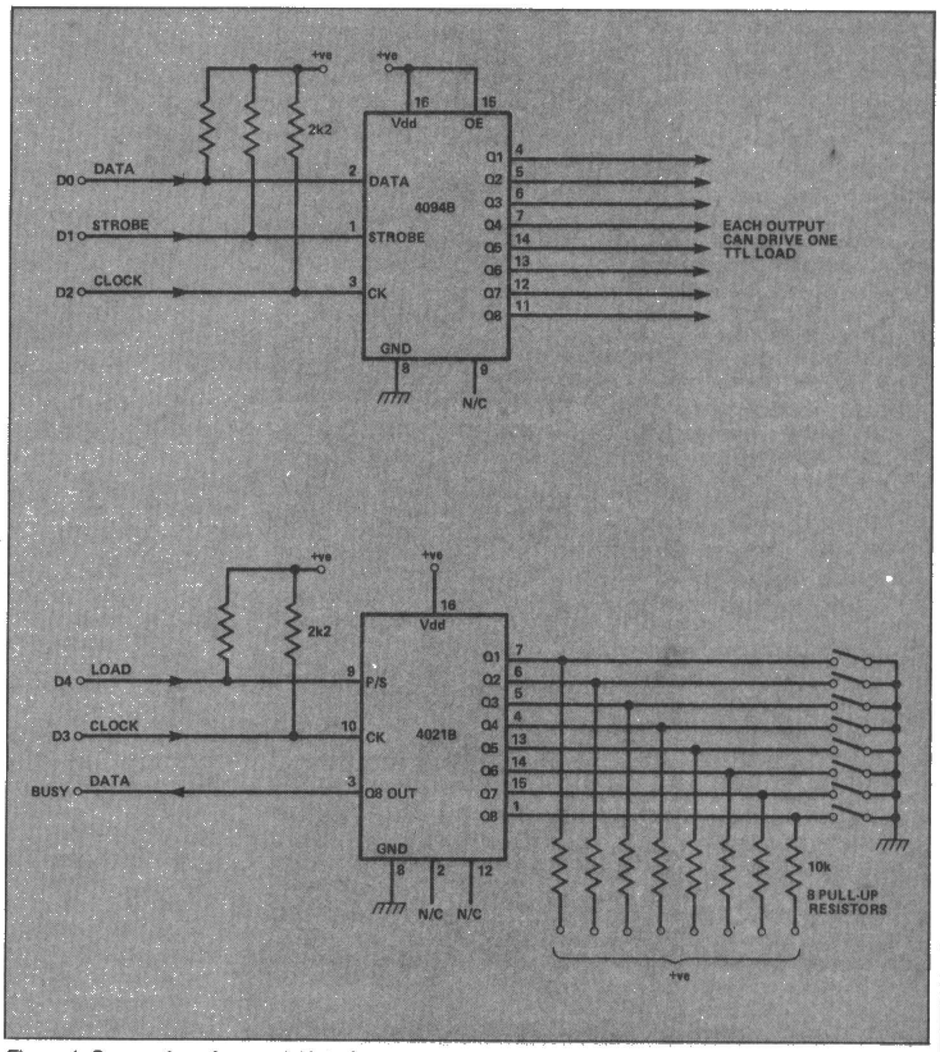


Figure 1. Suggestions for a serial interface.

LISTING 1.

```
708A                ;
708A                ;MACHINE CODE FOR "TWO MOTOR DRIVERS"
708A                ;EXAMPLE ONLY -- THIS IS NOT
708A                ;SPECIFIC TO ONE COMPUTER
7000                          ORG 7000H
7000                          LOAD 7000H
7000
7000
7000                ;ENTER HERE FOR MOTOR Y FORWARDS
7000 0E04           YFOR      LD C,4            ;"00000100"
7002 180C                     JR MAINCODE
```

would be responsible for extracting information from the 4021, to assemble the 1 byte representing the status of the buggy's switch sensors.

This 2-chip solution might appeal to you, but remember that the 4021 can only pass logical data. In the next circuit, **Figure 2,** the sensor-multiplexer can handle analogue data, which is always a useful facility to build into any design destined to be used in robotics.

CIRCUIT TWO

This circuit of five ICs offers a .umber of advantages over circuit one. The output latches are robust TTL, the input multiplexer is the massive 16 input CD4067B, and the whole design can, at a pinch, be run directly from Basic – a facility useful for testing and setting-up purposes.

Purists will note tht the 74LS174 latches are being under-used with only 5 of their 6 lines connected to the parallel data port. This is because the design is pared down to fit minimum-spec system. Note that it is assumed that only 8 parallel lines can be squeezed out of the computer, the eighth line coming from either D7 or the Strobe. Some computers have D7 **and** a strobe (which can sometimes be reached via machine code), and if this is the case with your machine, you should take advantage of the fact, and rewire to include the sixth bit of each 74LS174 in the new design.

Because of a happy chance in the design of the CMOS 4555 decoder (which is why, of course, we are using CMOS here), connecting Centronics STROBE to pin 15 enables you to drive the 74LS174 outputs using humble LPRINT commands, should you wish to do so!

SOME SOFTWARE

The action of circuit 2 is best seen from the viewpoint of a specific command instruction. Let us suppose that half of the "Two Motor Driver" circuit *(E&CM August p56)* is attached to IC2, with latched output L2-0 controlling motor ON/OFF and output L2-1 controlling motor direction. To drive the motor forward the now familiar byte 00000001 is required, but it must be sent to IC2. Before it is sent, consider the state of IC4. STROBE is off, so it's HIGH. IC4 is thus disabled and all outputs are LOW. All latches, ICs 1-3, are holding their previous data. To latch IC2, IC4 must put a HIGH out on its Q1 and to do this is must be given 01 on its A B inputs. Therefore 00100000 is ORed to 00000001 to give the byte 00100001 which is sent out as LPRINT CHR$(33). The automatic LPRINT low-going strobe enables IC4 which outputs a high-going strobe of identical duration to IC2 and the LPRINT data passes through to its proper destination.

LPRINT CHR$(1024+C) selects a sense line prior to reading it in. C is the address of the sense line, in the range 0 to 15. Remember that if you intend to use Centronics' BUSY as the IN port, then you will not be able to use LPRINT commands. LPRINT, when used to drive robots, needs BUSY to be tied to ground. It's time to look

LISTING 1 – Continued

```
7002
7002                ;ENTER HERE FOR MOTOR Y REVERSE
7004 0E0C    YREV        LD C,0CH       ;"00001100"
7006 1808                JR MAINCODE
7006
7006                ;ENTER HERE FOR MOTOR X FORWARDS
7008 0E01    XFOR        LD C,1         ;"00000001"
700A 1804                JR MAINCODE
700A
700A                ;ENTER HERE FOR MOTOR X REVERSE
700C 0E03    XREV        LD C,3         ;"00000011"
700E 1800                JR MAINCODE
700E
7010 3A3070  MAINCODE    LD A,(VALUE)   ;A DELAY VALUE
7013 47                  LD B,A         ;IS OBTAINED AND
7013                                    ;SAVED FOR LATER
7014 79                  LD A,C         ;MOTOR NUMBER AND
7014                                    ;DIRECTION INTO A
7015 C5      DELAYLOOP   PUSH BC
7016 F5                  PUSH AF
7017 CD3170              CALL OUTBYTE   ;TURNS ON MOTOR
7017
7017                ;NOW AN INBUILT DELAY OF 800H
7017                ;TIMES ROUND THE MINDEL LOOP
7017                ;TRY DIFFERENT VALUES FOR
7017                ;DIFFERENT TYPES OF MOTOR
7017
701A 010008              LD BC,0800H
701D 0B      MINDEL      DEC BC
701E 79                  LD A,C
701F B0                  OR B
7020 20FB                JR NZ MINDEL
7020                     ;MINIMUM DELAY FINISHED
7020
7022 C1                  POP BC
7023 78                  LD A,B
7024 00                  NOP            ;CALL YOUR "WRITE
7025 00                  NOP            ;ACCUMULATOR TO SCREEN"
7026 00                  NOP            ;ROUTINE HERE AND WATCH
7026                                    ;"VALUE" DECREASE
7026
7027 F1                  POP AF
7028 10EB                DJNZ DELAYLOOP
7028
702A 3E00                LD A,0         ;MAIN DELAY FINISHED
702C CD3170              CALL OUTBYTE   ;SO MOTORS OFF
702F C9                  RET
7030 00      VALUE       DB 0
7030
7031 C5      OUTBYTE     PUSH BC        ;THIS SIMPLE
7032 010500              LD BC,5        ;OUTBYTE ROUTINE DOES
7035 ED79                OUT (C),A      ;NOTHING EXCEPT PUT
7037 C1                  POP BC         ;THE ACCUMULATOR OUT
7038 C9                  RET            ;TO PORT NUMBER 5
7038                                    ;YOUR PORT ADDRESS MUST
7038                                    ;BE LOADED INTO BC
7038
7038                ;THIS TEST CALLS THE PREVIOUS ROUTINES
7038                ;KEYBOARD PRESSES FIRST LOAD "VALUE"
7038                ;AND THEN SELECT MOTOR OPTION
7038
7039 00      TEST        NOP            ;CALL YOUR
703A 00                  NOP            ;PIO INITIALISATION
703B 00                  NOP            ;IF NECESSARY
703B
703C 00                  NOP            ;CALL YOUR ROM AND
703D 00                  NOP            ;GET A KEYPRESS
703E 00                  NOP            ;INTO THE ACCUMULATOR
703E
703F FE30                CP "0"         ;IF "0" THEN
7041 CA0000              JP Z 0         ;ABORT TO MONITOR
7041
7044 FE31                CP "1"         ;IF "1" THEN USE
7046 2803                JR Z T1        ;PREVIOUS "VALUE"
7046
7048 323070              LD (VALUE),A   ;ELSE MAKE VALUE
7048                                    ;OF KEY THE NEW VALUE
7048
704B 013970  T1          LD BC,TEST     ;SOMEWHERE TO RETURN
704E C5                  PUSH BC        ;IS PUT ONTO STACK
704E
704F 00      T2          NOP            ;NOW LOOK
7050 00                  NOP            ;AT YOUR
7051 00                  NOP            ;KEYBOARD AGAIN
7052 FE31                CP "1"         ;X MOTOR FORWARD
7054 CA0870              JP Z XFOR
7057 FE32                CP "2"         ;X MOTOR REVERSE
7059 CA0C70              JP Z XREV
705C FE33                CP "3"         ;Y MOTOR FORWARD
705E CA0070              JP Z YFOR
```

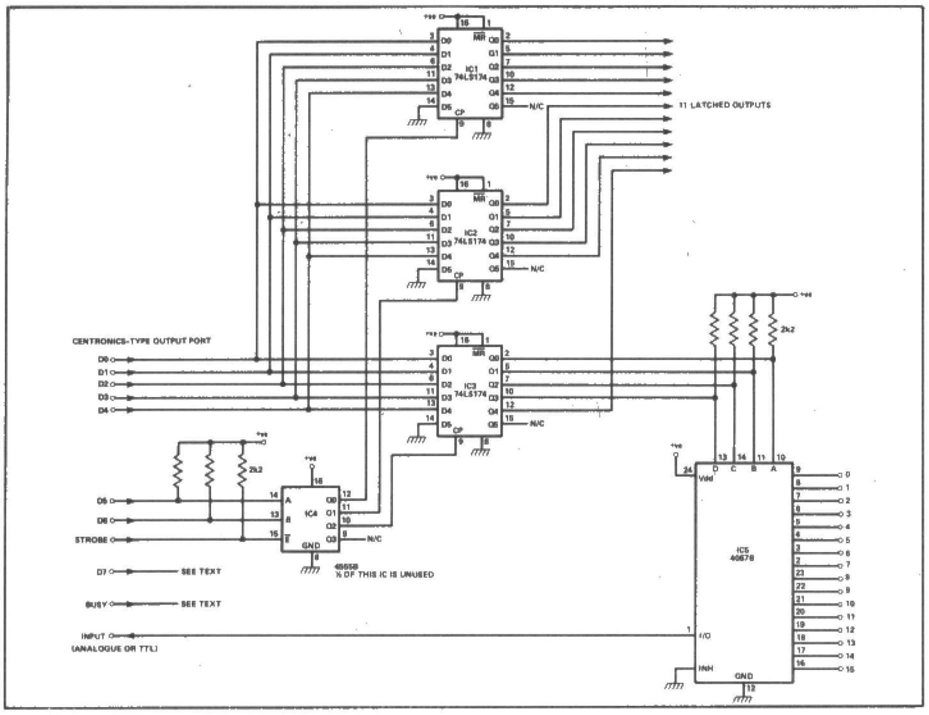


Figure 2. This sensor-multiplexer can handle analogue data.

at machine code, and we'll begin with that long overdue routine which drives the "Two Motor Drivers" board. **Listing 1** shows the source code, suitable for virtually any Z80 CPU computer. Use it to drive Wall Builder, Laurel or Hardy or any other dual motor device. It's an experimental program designed to assess gearing on different designs of motor, but it isn't set to work on the multiplexed circuits discussed this month, without due attention to masking in the OUTBYTE section of the code.

KITCHENWARE SENSORS

LEGO robotics is a low-cost robotics venture, and so every expense has been

"Every expense has been spared in fitting ESP to Hardy".

spared when it comes to fitting Hardy with extra sensory perception! To take advantage of all the new input lines, Hardy has been fitted with no fewer than eight bump-sensors representing eight sections of the compass rose. You will find, if you have endeavoured to build Hardy close to the original chunky shape, that a plastic bowl or dome will fit nicely over the Buggy's entire bodywork. This has the advantage of hiding a multitude of constructional errors (!), but also provides the basis for eight cheap sensors. A search for a suitable dome takes us away from LEGO, of course, but not for long . . . A 50p plastic sieve seemed to have the precise colour/weight/size requirements, but feel free to indulge yourself! Your choice must, however, be capable of being pivoted on the top of Hardy, and be able to move on its pivot whenever it hits an obstacle.

Figure 3 shows the idea. The dome is only slightly modified. A hole (A) is drilled to accept a LEGO axle (B);. Lines (C) and (D) represent the sides of a hollow cylinder – it could be part of a metal can, or more simply, a cardboard cylinder lined with kitchen foil. It is firmly glued or screwed to the dome at (E). Structure (F) is a solid cylinder (a piece of dowel rod is ideal) with eight drawing pins embedded in it. By

. . . source code is suitable for any Z80 computer . . .

selecting your hollow cylinder and solid cylinder sizes carefully, the metal of the former will only make contact with the drawing pins of the latter when the buggy bumps into an obstacle. By extending axle (B) into rod (G) a shepherd's crook arrangement can keep the wires (and there are nine from the homemade sensors themselves) reasonably free of the dome. If you intend to use "sensory Hardy" in rough environ-

LISTING 1 – Continued

```
7061 FE34                 CP "4"          ;Y MOTOR REVERSE
7063 CA0470               JP Z YREV
7066 18E7                 JR T2
7066
7066           ;PIO INITIALISATION (EXAMPLE)
7066           ;USING Z80 PIO CHIP
7068 C5        PIO        PUSH BC
7069 3ECF                 LD A,0CFH
706B 010600               LD BC,6
706E ED79                 OUT (C),A
7070 3EFD                 LD A,0FDH
7072 D306                 OUT (6),A
7074 3E0F                 LD A,0FH
7076 010700               LD BC,7
7079 ED79                 OUT (C),A
707B C1                   POP BC
707C C9                   RET
707C
707C
707C           ;NOW THE ACTUAL INITIALISATION
707D CD6870    INIT       CALL PIO        ;SET PIO
7080 3E00                 LD A,0
7082 CD3170               CALL OUTBYTE    ;MOTORS OFF
7085 3E01                 LD A,1
7087 323070               LD (VALUE),A    ;VALUE IS LOW
708A C9                   RET
```

LISTING 2.

```
7137           ;MACHINE CODE FOR READING 4067 CHIP
7137           ;EXAMPLE ONLY -- THIS IS NOT
7137           ;SPECIFIC TO ONE COMPUTER
7100                      ORG 7100H
7100                      LOAD 7100H
7100
0000           STRBIT     EQU 0           ;OR WHICHEVER BIT NEEDED
0005           STRPORT    EQU 5           ;OR WHICHEVER ADDRESS NEEDED
7031           OUTBYTE    EQU 7031H       ;OR WHEREVER IT IS
7031
7031
7100 3A3671    ENTERHERE LD A,(LATCH3)    ;PREVIOUS STATUS IC3
7103 E690                 AND 90H         ;BITS 7&4 ISOLATED
7105 F6C0                 OR 0C0H         ;IC 4 (4555) SET UP
7107 4F                   LD C,A          ;ALL STORED IN C
7107
7108 060F                 LD B,15         ;15 ADDRESSES TO READ
710A 213771               LD HL,RAMSTORE;SOMEWHERE TO PUT RESULTS
710D 79        READLOOP   LD A,C
710E B0                   OR B
710F CD3170               CALL OUTBYTE    ;PRESENT ADDRESS TO IC3
710F
710F                      ;NOW EITHER CALL STROBE
710F                      ;WITH "CALL STROBE"
710F                      ;OR ELSE TOGGLE D7 LOW
710F                      ;WITH "CALL TOGGLE"
```

LISTING 2 – Continued

```
710F                        ;DEPENDING ON HOW YOU
710F                        ;ARE ENABLING IC4
710F
710F
710F                        ;NOW EITHER READ BUSY
710F                        ;WITH, FOR EXAMPLE :-
710F             ;PUSH BC
710F             ;LD BC,BUSYPORT
710F             ;IN A,(C)
710F             ;POP BC
710F                        ;OR ELSE CALL AN A-to-D
710F                        ;CONVERTER, RETURNING
710F                        ;WITH VALUE IN ACCUMULATOR
710F
7112 77                     LD (HL),A       ;STORE SENSED DATA
7113 23                     INC HL
7114 10F7                   DJNZ READLOOP
7116 C9                     RET             ;ALL 16 SENSE LINES READ
7116
7117 C5          STROBE     PUSH BC
7118 010500                 LD BC,STRPORT
711B CBB7                   RES STRBIT,A
711D ED79                   OUT (C),A       ;SEND OUT THE LOW
711F 00                     NOP             ;WAIT A
7120 00                     NOP             ;WHILE
7121 00                     NOP
7122 CBC7                   SET STRBIT,A
7124 ED79                   OUT (C),A       ;SEND OUT HIGH AGAIN
7126 C1                     POP BC
7127 C9                     RET
7127
7128 CBBF        TOGGLE     RES 7,A         ;TAKE D7 LOW
712A CD3170                 CALL OUTBYTE    ;SEND THE LOW
712D 00                     NOP             ;WAIT A
712E 00                     NOP             ;WHILE
712F 00                     NOP
7130 CBFF                   SET 7,A         ;TAKE D7 HIGH AGAIN
7132 CD3170                 CALL OUTBYTE    ;SEND THE HIGH
7135 C9                     RET
7135
7136             LATCH3     DS 1
7137             RAMSTORE   DS 16
7137
```

ments then you must either be prepared for the dome to come adrift frequently, or you must glue some of the upper LEGO structures together using a strong epoxy glue.

Listing 2 shows how the sensors can be read in machine code, using the circuit containing the 4067 analogue multiplexer.

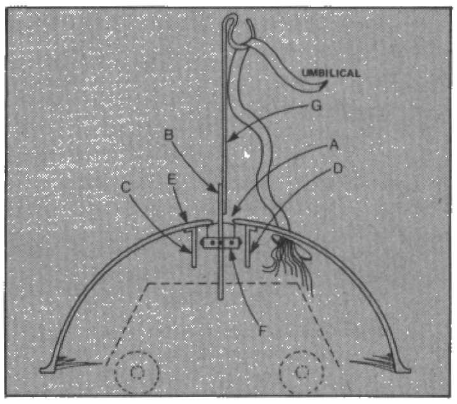


Figure 3. A plastic seive completeshardy's bodywork.

PARTS LIST

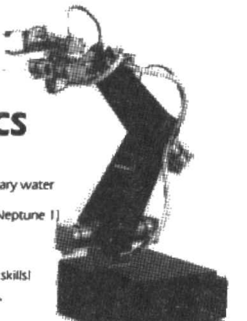
for the multiplexed parallel board.

IC1,2,3	74LS174
IC4	4555B
IC5	4067B
R1-7	2K2 0.25W
C1-4	100nF Disc ceramic

5 IC sockets; Veroboard.

A LEGO parts list for the Hardy buggy is available on request. Send a SAE to *E&CM.*

neptune



for low-cost training in

real-life robotics

The advanced design of the Neptune 2 makes it the lowest cost real-life industrial robot.

It is electro-hydraulically powered, using a revolutionary water based system (no messy hydraulic oil!)

It performs 7 servo-controlled axis movements (6 on Neptune 1) – more than any other robot under £10,000.

Its program length is limited only by the memory of your computer.

Think what that can do for your BASIC programming skills!

And it's British designed, British made.

Other features include:

Leakproof, frictionless rolling diaphragm seals.
Buffered and latched versatile interface for BBC VIC 20 and Spectrum computers.
12 bit control system (8 on Neptune 1).
Special circuitry for initial compensation.
Rack and pinion cylinder couplings for wide angular movements.
Automatic triple speed control on Neptune 2 for accurate 'homing in'.
Easy access for servicing and viewing of working parts.
Powerful – lifts 2.5 kg. with ease
Hand held simulator for processing (requires ADC option)

Neptune robots are sold in kit form as follows.

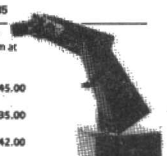
Neptune 1 robot kit (inc. power supply)	£1250.00	ADC option (components fit to main control board)	£95.00
Neptune 1 control electronics (ready built)	£295.00	Hydraulic power pack (ready assembled)	£435.00
Neptune 1 simulator	£45.00	Gripper sensor	£37.50
		Optional extra three fingered gripper	£75.00
Neptune 2 robot kit (inc. power supply)	£1725.00	BBC connector lead	£12.50
Neptune 2 control electronics (ready built)	£475.00	Commodore VIC 20 connector lead and plug-in board	£14.50
Neptune 2 simulator	52.00	Sinclair ZX Spectrum connector lead	£15.00

All prices are exclusive of VAT and valid until the end of March 1985

mentor
desk-top robot

This compact, electrically powered training robot has 6 axes of movement, simultaneously servo- controlled. It gives smooth operation, and its rugged construction makes it ideal for use in educational establishments. Other features include long-life bronze and nylon bearings, integral control electronics and power supply, special circuitry for inertial compensation, optional on-board ADC, and hand-held simulator as the teaching pendant. Like Neptune, Mentor's program length is limited only by your computer's memory. Programming is in BASIC.

Mentor is all-British in design and manufacture and comes in kit form at an astonishingly low price:



Mentor robot kit (inc. power supply)	£345.00
Mentor Control electronics (ready built)	£135.00
Mentor Simulator (requires ADC option)	£42.00
ADC option (Components fit to control electronics board)	£19.50
BBC connector lead	£12.50
Commodore VIC 20 connector lead and plug-in board	£14.50
Sinclair ZX Spectrum connector lead	£15.00

All prices are exclusive of VAT and valid until the end of March 1985

Cybernetic Applications

CYBERNETIC APPLICATIONS LIMITED
PORTWAY TRADING ESTATE, ANDOVER, HANTS SP10 3YR
TEL (0264) 50093 TELEX: 477019

OGRE 1

Your Robot (May 84)
"Its stylish good looks make an immediate impression"
"Will be the best buy around"
"Lifted a total weight of 4lbs!"

What Micro? (June 84)
"Is quite powerful and will life a load of 7lbs"
"Will sell for £225 rather than better known ones that cost nearer £600."

Computer Answers (July/August '84)
"The arm is very powerful with a bone-crushing grip. Constructionwise, Ogre is a real monster, the elbow rotates to 60, making it possible to grab any object of any height! A well made piece of machinery that should prove useful to both college and industry."

Incredible value for £195+VAT

Please allow 28 days delivery due to very heavy demand.

VIC20/CMB64, DC motor/stepper motor drive interface, fully independent stop/start/forward/reverse by simple pokes!!!

2 DC motor/1 stepper interface	£25.00
4 DC motor/2 stepper interface	£35.00
Interface Board for Ogre	£40+VAT

Complete with leads and software.

L.W. STAINES & CO
UNIT 2 RODING TRADING ESTATE
LONDON ROAD BARKING ESSEX IG11 8BU
01-591 2900

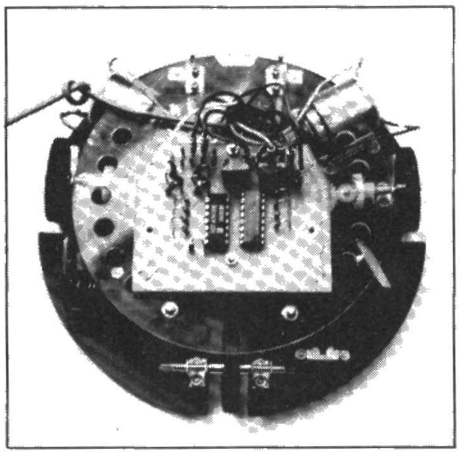
COMPUTER CONTROL FOR MOVITS

Ken Alexander reviews the new DEW interface which lets your computer control your movit.

At its simplest, a turtle is little more than a couple of motors mounted on a base and connected to a computer via a wire link. The majority of commercially available turtles however take this basic definition many stages further by building in all sorts of sophistications.

Unfortunately all of these extras cost money – the reason why many designs leave little change from £200. There would seem to be a market for a basic low cost turtle that offers the bare minimum of facilities yet also makes provision for future expansion. It is this market that the DEW interface/Line Tracer Movit combination is aimed.

The Line Tracer Movit is one of a range of low cost DIY robots marketed in this country by Prism. The Line Tracer Movit consists of a base on which are mounted two drive motors, a sensor board consisting of an infra red LED and two photo diodes. Some control electronics and various nuts and bolts hold everything together.



As it stands however, the Movit makes no provision for computer control. It is this omission that the DEW interface makes good.

EASY GOING

Assuming that the Line Tracer has been built according to the supplied instructions

TECHNICAL EXPLANATION

To the programmer, the DEW interface appears as a serial array of functions that may be sequentially selected by a series of high-low-high pulses transmitted via one of three wires linking the board to the computer. To the electronics engineer the interface consists of a 4017 decade/divider, a 4093 quad two input NAND gate, a pair of V-FET drivers and a handful of assorted resistors and diodes.

The reset input of the 4017 is taken directly to the 'touch resets' of the PCB while the clock input is connected, via the umbilical cord, to the output line of the computer designated as the serial output channel.

Resetting the 4017 will force the 0 pin of the IC to the low state and subsequent high-low-high transitions on the clock line will, in sequence, take lines 1 to 9 low. With output 9 low, the next clock pulse will return the IC to its initial state, ie line 0 low. The outputs can thus be considered as a ring, indeed this curcuit configuration is often known as a ring counter.

Table 1 shows the way in which the 10 lines of the 4017 have been assigned. In the case of the motor channels, the output of the counter IC is taken directly to the gate of one of the two V-FET devices (or both in the case of the left and right command). The user defined channels are direct connections to one of the 4017's output lines while the four input channels are taken to one of the gates in the quad NAND package. The output of these gates are selected by the 4017 in conjunction with a series of diodes.

TABLE 1

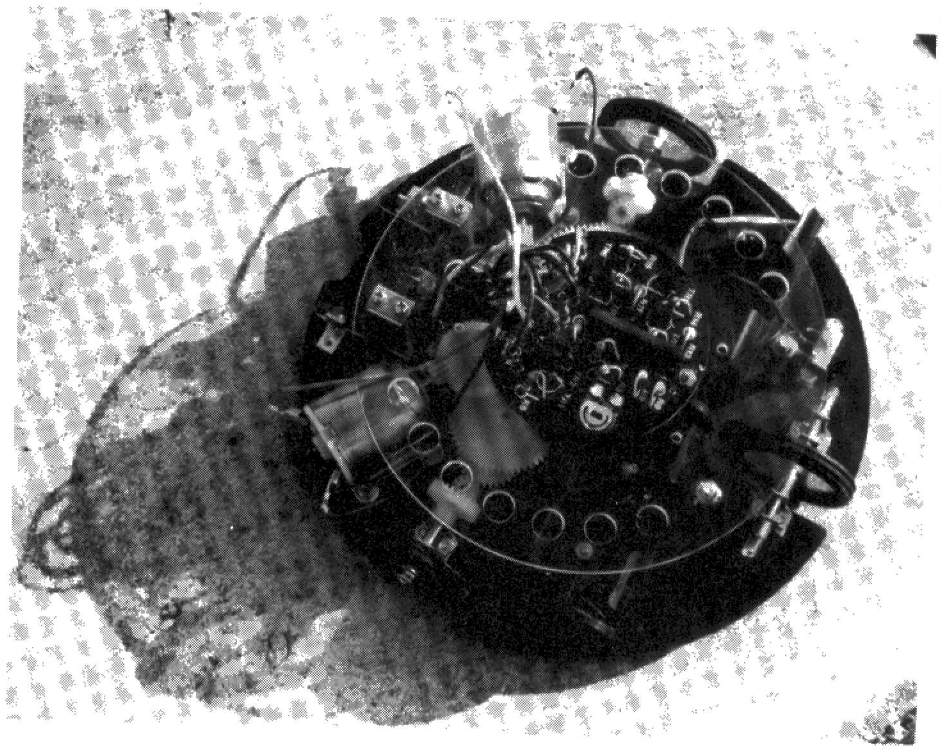
Channel No.	Function
0	User defined
1	Motor right
2	Motors left and right
3	Motor left
4	User defined
5	User defined
6	Sensor input
7	Sensor input
8	Sensor input
9	Sensor input

Although the hardware is of a fairly simple design it opens up a number of interesting possibilities. One of these is the prospect of controlling the speed of the drive motors via pulse width modulation. With quite straightforward software it would be possible to vary the duty cycle of the pulse appearing on a specified motor control line, thus varying the speed at which the associated motor is driven. The sensor inputs (two of which are tied high and two low) may be used in conjunction with the diodes that form the line tracer's detector assembly either in their 'Movit' role as, perhaps an intelligent line tracer, or they could be modified to provide some positional feedback in conjunction with a slotted disk mounted on the drive wheels. The other outputs could be used to provide inputs for tactile sensors. The three output lines could be used to directly drive LED's or, via a driver transistor, other devices such as a sound generator.

and that it is working satisfactorily, it is a simple matter to install the interface. Firstly the Line Tracer control PCB is removed and DEW's board mounted in its place. In keeping with Movit philosophy, there is no need to make any soldered connections to the interface board.

The board is connected to the controlling computer via a three wire (two plus screen) link rather than the common multi-way ribbon cable. For an explanation of the operation of the interface see the technical explanation section.

The interface supplied for review was designed for operation with the BBC micro and was supplied with two pieces of software. The first of these was written in BASIC and provided a simple robot driver, while the second, more sophisticated program was written in machine code. The latter program was fully annotated and study of this should provide users with a thorough understanding of the techniques used to control the interface.



. . . AND FINALLY

The combination of Movit Line Tracer and DEW interface allows users to explore the concepts of turtle operation at very low cost. The full price of the interface plus Movit is around £40, a fraction of the cost of other designs on the market. In its basic form the Movit turtle offers a very limited performance, but the system makes ample provision for expansion. Without spending a fortune it should be possible to provide some form of positional feedback and to build in tactile sensors in addition to other more esoteric detectors (perhaps for sound).

The interface is straightforward to control by means of software written in either Basic, or, for more demanding applications, in machine code. At present the interface is designed for operation with the BBC computer but if sufficient interest is shown, and surely there will be, DEW will offer versions for the Spectrum and Commodore 64.

The Movit interface costs £25 fully inclusive and is available from DEW Electronics at 21 Warwick Street, Oxford, OX4 1SZ.

HISOFT

C

for the ZXSpectrum

Hisoft is pleased to announce a new compiler for this popular and effective systems programming language. Not a tiny-C but an extensive, easy-to-use implementation of the language. Allows direct execution of compiled statements. Supplied with function library. Available direct from Hisoft for £25, or write for further details.

All prices, UK delivered, relate to 48K ZX Spectrum versions. Our software is available for many other Z80 machines e.g. Amstrad CPC 464, MSX, Memotech, SHARP MZ700, New-Brain, CP/M etc. Please write for details.

HISOFT

ULTRAKIT *£9·45*

The most powerful toolkit yet for ZX BASIC. All the features you will ever need; AUTO insert, full RENUMBER, block DELETE, CLOCK, ALARM, error trapping, break trapping. Full TRACE with single-step and much, much more. Makes ZX BASIC easy-to-use and powerful.

DEVPAC *£14*

An excellent assembler, an advanced line-editor, a comprehensive disassembler and a superb 'front panel' debugger all in one package. Used by many leading software houses to write their games. "Buy it!" Adam Denning 1984.

PASCAL *£25*

A powerful and almost full implementation of Pascal - not a Tiny Pascal. A valuable educational and development tool, programs typically run 40 times faster than a BASIC equivalent. Spectrum version includes Turtle Graphics package. " I haven't seen any other compiler that could match Hisoft's Pascal"

HISOFT

180 High Street North
Dunstable, Beds. LU6 1AT
Tel: (0582) 696421

64 character Spectrum printout

J. H. Woodhead shows you how to print out 64 Spectrum characters per line, or if you want to shout, in double and quadruple size.

This machine code routine enables the Spectrum to print either 64 characters per line or double size or quadruple size characters.

By opening two extra print channels, the routines can be integrated into Basic quite easily and you don't need to understand machine code.

Normal printing is done using channel 2, but another channel can be used with the hash symbol #. So if:

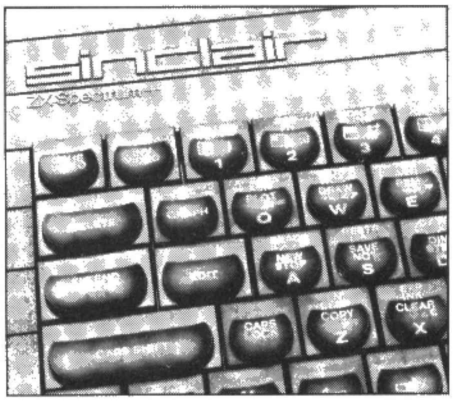
PRINT#8;"ABC"

is inputted, the print command is redirected to our own routine and ABC is printed double size. Alternatively:-

PRINT#9;"ABC"

will print 64 characters per line.

All sizes of characters can be put on the screen at the same time.


Software compatible with Spectrum+.

Before you do anything else you should type in CLEAR 64732 to reserve some memory above ramtop, then input the series of numbers in **Listing 1,** which is the actual printing routine and is 634 bytes long. The short program below should help you poke the numbers into the right place.

```
10  FOR A + 64733 TO 65368
20  INPUT P
30  POKE A, P
40  PRINT A, P
50  NEXT A
```

When this is done you should save it on cassette by typing in

SAVE "CHARS" CODE 64733,634

then, when it is needed for future use, simply type in

CLEAR 64732
LOAD "" CODE

The only thing that remains to be done now is to set up some vectors in the channel information area.

LISTING 1. Printing routine.

```
65033  205  254  13   6    3        65193  4    170  170  64   4
65038  33   160  80   209  62       65198  68   68   64   4    162
65043  170  15   50   19   254      65203  104  224  4    162  98
65048  245  197  229  6    4        65208  224  2    106  242  32
65053  48   31   126  230  15       65213  14   142  42   64   6
65058  119  26   230  240  182      65218  142  170  64   14   36
65063  119  36   126  230  15       65223  72   128  4    170  234
65068  119  26   203  39   203      65228  164  4    170  98   192
65073  39   203  39   203  39       65233  0    64   4    0    0
65078  182  119  36   19   16       65238  0    64   72   1    36
65083  227  24   29   126  230      65243  132  32   0    14   14
65088  240  119  26   203  63       65248  0    8    66   36   128
65093  203  63   203  63   203      65253  4    162  68   4    6
65098  63   182  119  36   126      65258  68   228  224  4    170
65103  230  240  119  26   230      65263  234  160  12   172  170
65108  15   182  119  36   19       65268  192  4    168  138  64
65113  16   227  37   205  219      65273  12   170  170  192  14
65118  11   225  193  241  56       65278  140  136  224  14   140
65123  2    35   13   195  232      65283  136  128  4    168  186
65128  10   0    0    0    0        65288  96   10   170  234  160
65133  4    68   64   64   90       65293  14   68   68   224  7
65138  0    0    0    10   234      65298  34   42   96   8    172
65143  174  160  70   204  102      65303  202  160  8    136  136
65148  196  21   18   72   168      65308  224  9    249  153  144
65153  0    0    0    0    68       65313  10   238  238  160  14
65158  0    0    0    2    68       65318  170  170  224  14   170
65163  68   32   4    34   34       65323  200  128  4    170  170
65168  64   10   70   196  160      65328  97   14   170  202  160
65173  0    68   228  64   0        64733  254  16   218  244  9
65178  0    4    72   0    0        64738  254  32   48   9    33
65183  224  0    0    0    4        64743  187  92   34   81   92
65188  64   1    18   72   128      64748  195  244  9    33   129
```

For this, just type in

POKE 23592, 125
POKE 23593, 255
POKE 23602, 194
POKE 23603, 253
POKE 23681, 2
POKE 23590, 127
POKE 23591, 255
POKE 23604, 221
POKE 23605, 252

Now the new routine is ready for use.

When running, the normal print statements are unaffected, but when you want to print 64 characters per line, use channel 9; for example:

PRINT#9;"ABC"

You can also LIST#9

Note that it only prints ASCII codes 32 to 90, so will not print lower case or graphics characters.

To print double or quadruple size use channel 8, eg:

PRINT#8;"AB"

This will print all characters except block graphics. To switch from one size to the other I have used the first two block graphics, code 129 and code 130, as control codes, so we can say

PRINT#8;"AB▝AND▘CD" to swap around

Scroll is a bit different, because it is not possible to print the scroll? message at the bottom of the screen without changing channels. So if the screen is full and requires scrolling up, it will just wait for a key to be pressed without the scroll? prompt.

For those who like to actually understand what they are doing, here is an explanation.

The offset for each channel is held in the system variable STRMS starting at 23568. The first six locations are used internally by the Spectrum then each channel offset occupies two bytes, so for #8 the offset will be at address 23568+6+2x8=23590.

The offset number is worked out from the start of channel information area at 23733, to where we put our vector to the print routine. Therefore if 23733 is added to the offset, we get the address of the vector.

In this case I put the vectors in some vacant system variables at 23604 but they could be placed in a REM statement in the BASIC program area.

Vectors should not be put above ramtop: they will not be safe. If your offset is a minus value, as in this case, then add 65536 to get a positive value.

The vector is the two byte address of our machine code routine.

Listing 2 is a disassembly of the print routine, and this is what it does:

On entry the accumulator holds the ASCII code to the character to be printed. First, the printable characters have to be sorted out from the position and colour controls, which are re-directed to the normal print routine.

FD18 the address in ROM of the character is found, which is then moved to the printer buffer.

FD1E the address in the display file is found and the screen is scrolled when required.

FD4C the character is now printed in either double or quadruple size, according to a flag in system variables at 5C81. This routine ends at FDC1. It is relocatable so you can move it somewhere else if you change the vectors.

The 64 character per line routine starts at FDC2.

First the printable characters are sorted out from the position and colour controls.

FDE1 The address of the character is found.

FDF4 Check for scroll.

FE1D A choice is made to print on the left or right hand side of the character square.

Then finally returning via ROM routine at øAE8 which stores the new print position. The small size character definitions are held at FE69 onwards.

For anyone with a 16K Spectrum, the following alterations should be made.

CLEAR 31964
Input the code to 31965 onwards
POKE 23603, 125
POKE 23605, 124
POKE 32229, 126
POKE 32279, 126

An explanation of the disassembly of the print routine for this project is shown above. However, readers who wish to see this in more detail may obtain a copy of the listing from *Electronics and Computing Monthly,* Spectrum 64 Character Screen Project, Scriptor Court, 155 Farringdon Road, London EC1R 3AD.

LISTING 1 – Continued

```
64753   92   254  129  32   3
64758   54   4    201  254  130
64763   32   3    54   2    201
64768   1    8    0    254  144
64773   56   8    237  91   123
64778   92   214  144  24   4
64783   237  91   54   92   111
64788   96   41   41   41   25
64793   17   0    91   237  176
64798   42   136  92   58   129
64803   92   245  189  56   14
64808   46   33   71   237  66
64813   229  68   205  155  14
64818   34   132  92   225  241
64823   245  60   60   79   124
64828   230  127  185  48   8
64833   229  205  254  13   225
64838   36   24   228  34   136
64843   92   241  71   8    237
64848   91   132  92   33   0
64853   91   197  8    71   8
64858   197  8    254  2    32
64863   37   8    6    4    175
64868   203  22   48   2    246
64873   192  203  22   48   2
64878   246  48   203  22   48
64883   2    246  12   203  22
64888   48   2    246  3    18
64893   20   18   20   35   16
64898   224  24   27   8    6
64903   2    203  22   159  230
64908   240  79   203  22   159

64913   230  15   177  0    18
64918   20   18   20   18   20
64923   18   20   35   16   232
64928   193  21   229  213  235
64933   37   205  219  11   209
64938   225  123  198  32   95
64943   62   0    138  230  248
64948   87   16   163  33   132
64953   92   52   46   136  53
64958   193  16   142  201  254
64963   96   32   2    62   64
64968   254  16   218  244  9
64973   254  32   48   9    33
64978   187  92   34   81   92
64983   195  244  9    254  91
64988   210  244  9    214  32
64993   135  135  33   105  254
64998   22   0    95   25   229
65003   42   132  92   237  75
65008   136  92   121  61   32
65013   6    14   33   5    205
65018   155  14   120  254  3
65023   48   16   8    58   4
65028   92   60   40   250  8
65333   4    168  66   192  14
65338   68   68   64   10   170
65343   170  64   10   170  164
65348   64   9    153  159  144
65353   10   164  74   160  10
65358   170  68   64   14   36
65363   72   224  221  240  254
65368   0
```

MAPLIN
1985 Buyer's Guide to Electronic Components
PRICES ON THE PAGE
Price £1·35
NOW WITH PRICES ON THE PAGE
including Heathkit section
PRICE £1·35
BUYER'S GUIDE TO ELECTRONICS AND HOME COMPUTERS
£1·25

The amazing Maplin Catalogue is here again! The new edition is packed with hundreds and hundreds of new electronic components to bring you right up to date with all the latest developments. As all home constructors agree (and a good many professionals too) the Maplin Catalogue is the one essential piece of equipment they really need. And now with all our **prices on the page** the Maplin Catalogue is better value than ever.

On Sale From 10th November 1984.

Pick up a copy as soon as it's published at any branch of W.H. Smith or in one of our shops. The price is still just £1.35, or £1.75 by post from our Rayleigh address (quote CA02C).

Post this coupon now for your copy of the 1985 catalogue.
Price £1.35 + 40p post and packing. If you live outside the U.K. send £2.40 or 11 International Reply Coupons.
I enclose £1.75.

Name ....

Address ....

....

.... ECM/12/84

MAPLIN
ELECTRONIC SUPPLIES LTD

Maplin Electronic Supplies Ltd. Mail Order: P.O. Box 3, Rayleigh, Essex SS6 8LR.
Tel: Southend (0702) 552911. •Shops at: 159-161 King Street, Hammersmith, London W6. Tel: 01-748 0926. •8 Oxford Road, Manchester. Tel: 061-236 0281.
•Lynton Square, Perry Barr, Birmingham. Tel: 021-356 7292.
•282-284 London Road, Westcliff-on-Sea, Essex. Tel: 0702-554000.
•46-48 Bevois Valley Road, Southampton. Tel: 0703-25831. All shops closed all day Monday.